Methods for Fast Linear Programming

Weston Jackson, Edo Roth, Flora Park

March 2017

Abstract

Linear programming helps us solve problems where we want to find the best outcome subject to linear constraints. Efficiently solving linear programs is central to a variety of important optimization algorithms. These problems include max-flow problems, scheduling problems, TSP problems, and many other algorithms that can be solved using positive definite linear systems. In this paper, we cover methods for fast linear programming, focusing predominantly on the evolution of interior point methods in the last thirty years. Specifically, we survey several papers in order to see how algorithms have evolved in addressing one of the most significant computational bottlenecks in interior point methods: the inverse maintenance problem. Then, we proceed to examine Lee and Sidford's algorithms for addressing this problem, which are known to be optimal in many circumstances.

Contents

1 Introduction

In this project we analyze papers by Karmakar, Renegar, Nesterov and Nemirovski, and lastly Vaidya to grasp how interior point methods have evolved, specifically, in addressing inverse matrix approximation. The papers we examine are: "A new polynomial-time algorithm for linear programming" by Karmakar, "Acceleration and parallelization of the path-following interior point method for a linearly constrained convex quadratic problem" by Nesterov and Nemirovski, "A polynomial-time algorithm, based on newton's method, for linear programming" by Renegar, and "Speeding-up linear programming using fast matrix multiplication" by Vaidya. By doing so, we build an understanding of the evolution of linear programming algorithms.

After giving historical background, we examine the publication by Lee and Sidford: "Efficient inverse maintenance and faster algorithms for linear programming." The paper provides an improved algorithm by reducing the computation cost for solving linear programs. Specifically, we examine how their work achieves inverse maintenance under the ℓ_2 stability assumption, leading to their main conclusion under the σ stability assumption. Finally, we also include a brief description of linear programming, descriptions of previous methods for solving linear programs, and Lee and Sidford' improvements to the path finding problem in the Appendix.

2 Interior Point Methods

Interior point methods traverse the interior of the feasible region before finding an ϵ-approximate solution. Nisheeth K. Vishnoi [1] gives the following as a general description of interior point methods:

- Start from some $x_0 \in K$
- Walk the *central path* in order to improve the objective at each iteration
- Stop when we have an ϵ-approximation of $c^T x^*$

Interior point methods generally rely on a several specific strategies in order to perform this optimization efficiently.

2.1 Newton's Method

One of the most important tools that interior point methods use to improve the objective at each iteration is *Newton's method*. Newton's method is used to find successively better approximations for the zeros of a real-valued function. In our case, we use Newton's method to find the root of the derivative of a convex function f, which allows us to approach the unique minimizer over a series of steps.

Definition 1. *A function f is convex if and only if*

$$f(\lambda x + (1-\lambda)\eta) \leq \lambda f(x) + (1-\lambda) f(y), \forall x, y \in \mathbb{R}^n, \lambda \in [0,1]$$

Assume we have query access to ∇f and $\nabla^2 f$. We define a *Newton step* given some f and x_k as follows:

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

where η is the step size. We can show that a Newton step improves the objective at each iteration by considering the second order approximation of f at x:

$$f(y) = f(x) + f(y-x)^T \nabla f(x) + \frac{1}{2}(y-x)^T \nabla^2 f(x)(y-x)$$

Let y^* be the minimizer of this function, taking the derivative with respect to y gives us the following minimization:

$$y^* = x - (\nabla^2 f(x))^{-1} \nabla f(x)$$

Thus, we can see that Newton's method uses the second derivative in order to make steps toward the minimum. Note that each Newton step requires us to find some intermediate y_k s.t. $y_k \nabla^2 f(x_k) = \eta \nabla f(x_k)$. Solving this linear system of equations at each iteration is a major computational bottleneck of the interior point method. Nonetheless, given certain conditions, each Newton step allows us to approach the minimum quadratically fast.

Theorem 2. *Let f be a convex real-valued function with gradient $\nabla^2 f(x)$, and let x^* be its minimum. Assume $\| \nabla^2 f(x)^{-1} \| \geq \frac{1}{h}$ and $\| \nabla^2 f(x) - \nabla^2 f(y) \| \leq L \| x - y \|$ holds for all points x, y at distance $\leq r$ from x^*. We have the following:*

$$\| x_1 - x^* \| \leq \frac{L}{2h} \| x_0 - x^* \|^2$$

where x_0 is the starting point, and x_1 is the Newton step from x_0.

This theorem implies that Newton's method converges in $O(\log \log \frac{\|x_0 - x^*\|}{\epsilon})$ time.

2.2 Barrier Functions

While Newton's method allows us to improve the objective on each iteration, because each Newton Step is an approximation, there is no guarantee that each iteration will remain in the feasible region. Thus, we need to manipulate the objective function in order to ensure that always remain in the polytope defined by the constraint matrix. To do so, interior point methods also rely on the use of a *barrier function*, which increases in value as the algorithm approaches a constraint hyperplane.

Definition 3. *A barrier function F if a function which satisfied the following properties:*

- $F(x) < \infty, \forall x \in K$
- $F(x) \to \infty$ *as x approaches any constraint in K*
- F *is strictly convex*

One example of the barrier function is the *logarithmic barrier* function:

$$F(x) = -\sum_{i=1}^{m} \log (b_i - a_i^T x)$$

where a_i is the ith constraint as a column vector. We notice that as we approach the boundary of the polytope, $a_i^T x - b_i = 0$, and thus the barrier function approaches infinity. Additionally, the logarithmic barrier function ensures we do not violate a constraint, as the log function is undefined when $a_i^T x - b_i < 0$. Note also that F is the sum of convex functions, and thus strictly convex.

With the barrier function F, we can perturb the objective function:

$$f_\eta(x) = \eta c^T x + F(x)$$

At each iteration, we can increase η and increase the influence of $c^T x$. Additionally, because $\nabla^2 f_\eta(x) = \nabla^2 F(x)$, we know that f_η is strictly convex and has a unique minimizer x_η^*. The set $\{x_\eta^* : \eta \geq 0\}$ is defined and continuous and is referred to as the central path. And as $\eta \to \infty$ we note that $f_\eta(x) \to f(x)$ and thus:

$$\lim_{\eta \to \infty} x_\eta^* = x^*$$

The generic interior point method is shown in Algorithm 1.

Algorithm 1 Interior Point Method

Start at $x_0 \approx x_0^*$
for $t = 1...T$ **do**
  $\eta_{t+1} = \eta_t(1 + \alpha)$ (for some α)
  $x_{\eta_{t+1}}^* = NewtonStep(f_{\eta_{t+1}}, x_{\eta_{t+1}})$
end for
return $x_{\eta_T}^*$

3 Historical Background

3.1 Overview

The first interior point method was developed in 1984 by Karmarkar. Contributions from Renegar, Nesterov and Nemirovski, and Vaidya improved his methods and soon developed into the interior point model from before.

Inverse matrix approximation is needed because each Newton step requires us to invert a Hessian matrix. Although this can be done in polynomial time, it is prohibitively expensive. Instead of explicitly calculating the Newton step, the run-time of interior point methods can be improved by maintaining an approximate inverse at each iteration. Assuming the inverse matrix does not change too much at each iteration, it suffices to perform simple computation in order to approximate the value of the inverse.

3.2 Karmarkar

In 1984, Karmarkar [2] announced the first polynomial time algorithm to solve linear programs using interior point methods. Let n be the number of variables and m be the number of constraints, and L be the number of bits in the input defined as:

$$L = \log(1 + det_{max}) + \log \rho + \log(m + n)$$

where ρ is the least common multiple of the denominators of all the numbers in the input. Karmarkar's algorithm traverses the feasible region in $O(mL)$ iterations, each time solving a simplified optimization problem that takes $O((n^{1.5}m+n^2)L)$ time. The algorithm works by first transforming the feasible polytope into a simplex, inscribing a sphere in the transformed space, and moving in the direction of steepest descent in the simpler transformed space.

At each iteration, Karmarkar's algorithm computes a projection which finds the direction that most quickly maximizes x. The algorithm then moves in the direction opposite of the projection in order minimize x. The major computational difficultly in Karmarkar's algorithm involves the use of this projection operator, which requires solving the following least squares problem:

$$c_p = (I - B^T(BB^T)^{-1}B)Dc$$

where $D \stackrel{\text{def}}{=} diag(x_1...x_n)$, c is the cost function, $B = [\frac{AD}{e^T}]$, A is the constraint matrix, and e^T is an all 1's row vector. The most difficult part of this least squares problem is computing the inverse of

$$BB^T = \begin{bmatrix} AD^2A^T & 0 \\ 0 & n \end{bmatrix}$$

Because A and n are fixed, this inverse only changes based on the diagonal matrix D. Let D and D' be two diagonal matrices. Karmarkar explains that if D and D' are similar, then $AD^2A^T \approx AD'^2A^T$. Additionally, if D and D' differ in one entry, we can perform $O(n^2)$ operations to compute AD'^2A^T given AD^2A^T. Then, assuming the diagonals do not change too much, Karmarkar then showed that he could perform rank-one updates to the inverted matrix $(A^TDA)^{-1}$ in order to maintain an approximation at each iteration.

The result was that Karmarkar avoided the $O(n^2m)$ operations normally needed for the projection operator by employing an approximate inverse matrix. In this way, the cost of his updates was approximately $O(n^2\sqrt{m}+nm)$ time, each to a precision of $O(L)$ bits. The use of an approximate inverse matrix was a novel way to reduce the run-time of his linear program solver, and would be improved in subsequent papers.

Karmarkar then proved his algorithm was optimal through the use of a potential function. At each iteration t he optimized the following modified objective

$$f(x_t) = \sum_i \ln(\frac{c^Tx_t}{x_i})$$

Note that Karmarkar's potential function soon developed into the more general notion of the barrier function used in all interior point methods. Karmarkar showed that this function was reduced by a constant at each iteration. After $\widetilde{O}(mq)$ steps, the algorithm returned a feasible point such that

$$c^Tx_t \leq c^Tx_0 2^{-q}$$

The algorithm must terminate if $c^Tx_t < 2^{-2L}c^Tx_0$, thus the total number of iterations is $O(mL)$ in the worst case. Note that at each iteration, while x_t does not necessarily improve the objective function, it is guaranteed to improve the potential function.

Finally, because Karmarkar's algorithm only needed $O(mL)$ iterations to converge, he achieved a total of of $\widetilde{O}((n^2\sqrt{m}+nm)mL)$ arithmetic operations, each to a precision of $O(L)$ bits.

3.3 Nesterov and Nemirovski

Nesterov and Nemirovski [3] improved the runtime of interior point methods in 1989. Focusing on the inverse matrix approximation bottleneck in Karmarkar's algorithm, Nesterov and Nemirovksi used conjugate gradient methods to speed up inverse matrix approximation.

Nesterov and Nemirovski noted that there were three ways in which interior point methods could be improved:

- Implementation of fast linear algebra algorithms instead of standard routines.
- Recursive computation of the approximate inverse Hessian matrix to perform Newton steps.
- Parallelization of computations.

Let $O(m^\omega)$ be the time it takes to compute a fast matrix multiplication of $m \times m$ matrices. Nesterov and Nemirovski noted that the basic implementation of Karmarkar's algorithm required $O(n^2 m)$ iterations. With fast matrix multiplication and inverse maintenance approximation, this could be improved to $\widetilde{O}(n^{\omega-1}\sqrt{m} + n^2 m^{0.5\omega-1} + nm) \approx \widetilde{O}(m^{0.5}n^{1.37} + nm + m^{0.19}n^2)$. Nesterov and Nemirovski built to improve Karmarkar's ideas with fast matrix multiplication.

They first noticed that Karmarkar's algorithm approximation procedure is unbalanced, in that it has a different runtime at each iteration to approximate the inverse. In order to improve the runtime, Nesterov and Nemirovski used the conjugate-gradient method, which using a symmetric matrix Q, could perform an even cruder approximatation to $(A^T DA)^{-1}$ at each iteration. By balancing the crudeness of the approximation with the decreased the complexity of updating the approximate inverse Hessians, Nestrov and Nemirovski achieved the following bound for one iteration of Karmarkar's algorithm:

$$\widetilde{O}(m^{0.5}n^{1.37} + m^{1.9} + m^{0.17}n^{1.9})$$

This improved on Karmarkar's algorithm when n and m are on the same order.

3.4 Renegar

Renegar's [4] interior point method in 1988 improved the path finding method of Karmarkar. He incorporated Newton's method and the log-barrier function into Karmarkar's algorithm.

Renegar chooses to focus on solving the objective $\max c \cdot x$ s.t. $Ax \geq b$. We define $\bar{A} = [\alpha_1, \ldots, \alpha_m, c, \ldots, c]^T$ where α_i denotes the original ith row of A, and $(b^{(0)})^T = (b_1, \ldots, b_m, k^{(0)}, \ldots, k^{(0)})$. Lastly, we define $(\bar{A}, b^{(0)})$ to be $\bar{A}x \geq b$. The Renegar algorithm is based on the idea of approximating the "center"s of a sequence of systems. The paper proves the following theorem inductively on j.

Theorem 4. *Let $\delta = \frac{1}{13}\sqrt{l}$ and # Newton = 1 and apply Newton's method with*

$$k^{opt} - c \cdot x^{(j)} \leq (1 - \frac{45l}{46(m+l)})(1 - \frac{\sqrt{l}}{14(m+l)})^j (k^{opt} - k^{(0)})$$

This led to an LP Solver that required only $O(\sqrt{m+n}L)$ iterations to converge.

3.5 Vaidya

In 1989, Vaidya [5] showed how to solve linear programs with $O((m+n)^{1.5}nL)$ operations, each to a precision of $O(L)$ bits. His algorithms makes use of fast matrix multiplication improved low-rank update formulas. Vaidya denotes M as the Hessian of any barrier function for an interior point method.

His algorithm gives three main ideas to reduce the number of operations needed in maintaining the inverse of matrix M:

- The first idea is to not explicitly calculate the inverse of M, but rather use a constant number of additions, subtractions and multiplications of matrices to get an expression for M^{-1}. Later, when M^{-1} is multiplied by a vector, this expression suffices.
- The second idea is to split this computation of the expression for M^{-1} into precomputations that use fast matrix multiplication, and incremental updates. This results in an algorithm with $O(\sqrt{m}L)$ total iterations split into periods with r iterations each. During each period, the necessary precomputations are first performed, consisting of explicitly recomputing M^{-1} (and some other related necessary matrices) using fast matrix multiplication. Then, the resulting expression for M^{-1} is gradually updated after each period using the precomputed matrices.
- Finally, they discuss a choice of period size r so that the number of operations for the precomputations and the number of operations for the incremental updates is balanced.

Using these ideas allows them to prove a bound of $O((m+n)^{1.5}nL)$ on the number of arithmetic operations performed by the algorithm, each to a precision to $O(L)$ bits.

4 Lee and Sidford's Inverse Maintenance Approximation

4.1 Overview

Lee and Sidford [6] further improve the efficiency of LP optimization, formally defining the *inverse matrix maintenance problem* for sparse constraint matrices. Because maintaining a matrix inverse is a common problem in interior point methods, Lee and Sidford focused on developing an algorithm that maintains an approximate matrix inverse. The formal problem is defined as follows:

Definition 5. *Let m be the number of constraints and n be the number of variables. Given a data matrix $A \in \mathbb{R}^{m \times n}$ and a number of rounds r, at each round k, we receive a non-negative vector $d^{(k)} \in \mathbb{R}^m_{>0}$. Output a $\mathcal{T}$-time solver $S^{(k)}$ for $A^T D^{(k)} A$ such that $\mathcal{T}$ and the time to construct $S^{(k)}$ are small.*

In this case, $D^{(k)} \in \mathbb{R}^{m \times m}$ is a diagonal matrix with $D^{(k)}_{ii} = d^{(k)}_i$. Now, let ω be the matrix multiplication constant and $\text{nnz}(A)$ be the number of non-zero entries of A. Lee and Sidford show how to solve the inverse matrix maintenance problem in $\widetilde{O}(\text{nnz}(A) + n^\omega)$ preprocessing time and $\widetilde{O}(\text{nnz}(A) + n^2)$ amortized time per round. This allows for solving a linear program in $\widetilde{O}((\text{nnz}(A) + n^2)\sqrt{n}\log(\epsilon^{-1}))$ time.

Their proof is broken into two parts. First, Lee and Sidford show that we can solve the inverse matrix maintenance problem under the ℓ_2 stability assumption. They then use this proof under the ℓ_2 to prove a stronger theorem under the σ-stability assumption.

4.2 Inverse Maintenance under the ℓ_2 stability assumption

The main theorem proved under the ℓ_2 assumption is as follows:

Theorem 6. *Suppose we are given matrix $A \in \mathbb{R}^{m \times n}$, a vector $b^{(0)} \in \mathbb{R}^n_{>0}$, a number of rounds $r > 0$, and in each round $k \in [r]$ we receive $d^{(k)} \in \mathbb{R}^m_{>0}$ such that $B^{(k)} = A^T D^{(k)} A$ is positive definite. Further, suppose that the number of pairs (i, k) such that $d_i^{(k)} \neq d_i^{(k-1)}$ is bounded by $\alpha \leq n$ and supposed that there is $\beta > 2$ such that $\beta^{-1} B^{(0)} \preceq B^{(k)} \preceq \beta B^{(0)}$. Then in round k, we can construct symmetric matrix K such that $K \approx_{O(1)} (B^{(k)})^{-1}$ and we can apply K to an arbitrary vector in time $\widetilde{O}(n^2 \log(\beta))$. Furthermore, the algorithm takes time $\widetilde{O}(sn^{\omega-1} + \alpha r n (\frac{\alpha}{r})^{\omega-2} + r\alpha^{\omega})$ where $s = \max\{nnz(d_0), n\}$.*

This improves on the running times of previous papers, which had an expensive additive $O(mn^{\omega-1})$ term. To prove this theorem, Lee and Sidford first use the notion of ℓ_2 stability.

Definition 7. *The inverse maintenance problem satisfies the ℓ_2 stability assumption if $\forall k \in [r]$, we have $\| \log d^{(k)} - \log d^{(k-1)} \|_2 \leq 0.1$ and $\beta^{-1} A^T D^{(0)} A \preceq A^T D^{(k)} A \preceq \beta A^T D^{(0)} A$ holds for $\beta = poly(m)$.*

Using the ℓ_2 stability assumption, we have that at most $O(r^2)$ coordinates of $d^{(k)}$ change by a fixed multiplicative constant in the r rounds of the inverse maintenance problem. Thus, the value of $d^{(k)}$ does not change by too much at each iteration. Lee and Sidford then compute an initial sparse approximation, or sparsifier, of $A^T D^{(0)} A$ using only $\widetilde{O}(n)$ rows of A in time $\widetilde{O}(\text{nnz}(A) + n^{\omega})$. At each iteration the algorithm, they use separate techniques to handle the rows from the sparsifier, $e^{(k)}$, and the other rows, $f^{(k)}$, where $d^{(k)} = e^{(k)} + f^{(k)}$.

Lee and Sidford maintain the sparsifier using the following Lemma:

Lemma 8. *Let $A \in \mathbb{R}^{m \times n}$, $d^{(0)} ... d^{(r)} \in \mathbb{R}^m_{>0}$, and $B^{(k)} \stackrel{def}{=} A^T D^{(k)} A$. Suppose that the number of pairs such that $d_i^{(k)} \neq d_i^{(k-1)}$ is bounded by $\alpha \leq n$. We can explicitly output $C \in \mathbb{R}^{m \times n}$ and $V^{(k)} \in \mathbb{R}^{m \times m}$ in each round k such that:*

$$(B^{(k)})^{-1} = (B^{(0)})^{-1} - C^T \Delta^{(k)} V^{(k)} \Delta^{(k)} C$$

where $\Delta^{(k)}$ is a $m \times m$ diagonal matrix with $\Delta^{(k)}_{ii} = d_i^{(k)} - d_i^{(0)}$. Furthermore, $V^{(k)}$ is an $\widetilde{O}(\alpha) \times \widetilde{O}(\alpha)$ matrix if we discard zero rows and columns. The total time per round is $O(mn^{\omega-1} + \alpha n r (\frac{\alpha}{r})^{\omega-2})$.

This lemma can be proved using fast matrix multiplication and the ℓ_2 stability assumption. Furthermore, Lee and Sidford also rely on subspace embeddings in order to approximate intermediate matrices. The main theorem, proved in a separate paper, is as follows:

Theorem 9. *There is a distribution $\mathcal{D}$ over $\widetilde{O}(n\epsilon^{-2}) \times m$ matrices such that for any $A \in \mathbb{R}^{m\times n}$, with high probability in n over the choice of $\Pi \sim \mathcal{D}$, we have $A^T\Pi^T\Pi A \approx_\epsilon A^TA$, where sampling from $\mathcal{D}$ and computing ΠA can be done in $\widetilde{O}(nnz(A))$ time.*

Lee and Sidford then show how to maintain the inverse $(B^{(k)})^{-1}$ at each iteration.

- Let $B^{(k)} = A^TD^{(k)}A$ be a positive definite matrix. Lee and Sidford note that $(B^{(k)})^{-1} = (A^TE^{(k)}A + A^TF^{(k)}A)^{-1}$, where $A^TE^{(k)}A$ is the matrix maintained by the sparsifier rows and $A^TF^{(k)}A$ is the matrix maintained by the new rows.
- Let $e^{(k)}$ be the rows used from the the initial sparsifier. Using the previous lemma, we can calculate the approximate contribution from $e^{(k)}$ in time $O(sn^{\omega-1} + \alpha nr(\frac{\alpha}{r})^{\omega-2})$. While some error is induced in order to ensure the calculated matrix $A^TE^{(k)}A$ is invertible, this only increases the error by an $O(1)$ factor. Thus, this allows us to maintains the inverted matrix $(A^TE^{(k)}A)^{-1}$.
- We can then use the Woodbury Matrix Identity to define

  $$(B^{(k)})^{-1} = (A^TE^{(k)}A)^{-1} - (A^TE^{(k)}A)^{-1}A^TF^{(k)}(M^{(k)})^{-1}F^{(k)}A^T(A^TE^{(k)}A)^{-1}$$

  where $M^{(k)} = ((F^{(k)})^\dagger)^{-1} + (N^{(k)})^TN^{(k)}$, and $N^{(k)} = (E^{(k)})^{1/2}A(A^TE^{(k)}A)^{-1}A^TF^{(k)}$.
- Calculating $N^{(k)}$ explicitly would be prohibitively expensive, thus Lee and Sidford use subspace embeddings to approximate $(N^{(k)})^TN^{(k)}$ with $(N^{(k)})^T\Pi^T\Pi N^{(k)}$.
- Maintaining $(N^{(k)})^T\Pi^T\Pi N^{(k)}$ is much cheaper, requiring only $\widetilde{O}(\alpha^\omega)$ extra time per-iteration. The total time for maintaining this matrix product is

  $$\widetilde{O}(\alpha nr(\frac{\alpha}{r})^{\omega-2} + sn^{\omega-1} + r\alpha^\omega)$$

  which gives us the improved run-time bound needed for the theorem.
- Finally, letting

  $$\Lambda = (A^TE^{(k)}A)^{-1}A^TF^{(k)}(M^{(k)})^{-1}F^{(k)}A^T(A^TE^{(k)}A)^{-1}$$

  Lee and Sidford are able to maintain a spectral approximation to $(B^{(k)})^{-1}$ such that

  $$(A^TE^{(k)}A)^{-1} - \Lambda^{(k)} \approx_{O(1)} (B^{(k)})^{-1}$$

Their proof then leads to the following result:

Theorem 10. *Suppose that the inverse maintenance problem satisfies the ℓ_2 stability assumption. Then there is an algorithm that maintains a $\widetilde{O}(nnz(A) + n^2)$-time solver with high probability in time $\widetilde{O}(sn^{\omega-1} + r(mn^{\omega-1})^{\frac{2\omega}{2\omega+1}})$, where $s = max\{max_{k\in[r]}nnz(d^{(k)}), n\}$, and r is the number of rounds.*

Algorithm 2 gives the entire algorithm explicitly.

Algorithm 2 Algorithm for the ℓ_2 Stability Assumption

Input: Initial point $d^{(0)} \in \mathbb{R}^m_{>0}$
Set $d^{(apr)} := d^{(0)}$
$Q^{(0)} \stackrel{\text{def}}{=} A^T D^{(apr)} A$
Let $K^{(0)}$ be an approximate inverse of $Q^{(0)}$ computed using 6
Output: A $\widetilde{O}(n^2 + \text{nnz}(A))$-time linear solver for $A^T D^{(0)} A$
for each round $k \in [r]$ **do**
  Input: current point $d^{(k)} \in \mathbb{R}^m_{>0}$
  for each coordinate $i \in [m]$ **do**
    if $0.9 d_i^{(apr)} \leq d_i^{(k)} \leq 1.1 d_i^{(apr)}$ is false **then**
    $d_i^{(apr)} := d_i^{(k)}$
  end for
  $Q^{(k)} \stackrel{\text{def}}{=} A^T D^{(apr)} A$
  Let $K^{(k)}$ be an approximate inverse of $Q^{(k)}$ computed using 6.
  Output: A $\widetilde{O}(d^2 + \text{nnz}(A))$-time linear solver for $A^T D^{(k)} A$
end for

4.3 Inverse Maintenance under the σ stability assumption

Lee and Sidford then show how to better approximate $A^T D^{(k)} A$ using the idea of *leverage scores* for matrices. The leverage score of a row $i \in [m]$ is $\sigma_i =: [A(A^T A)^{-1} A^T]_{ii}$. Leverage scores satisfy $\sigma_i \in [0, 1]$ and $\| \sigma \|_1 \leq n$. We use $\sigma_A(d)$ to denote the the leverage scores of matrix $D^{1/2} A$. Lee and Stanford then use the σ-stability assumption to assume that the leverage scores of the matrix are stable.

Definition 11. *The inverse maintenance problem satisfies the σ stability assumption if $\forall k \in [r]$, we have $\| \log d^{(k)} - \log d^{(k-1)} \|_{\sigma_A(d^k)} \leq 0.1$, $\| \log d^{(k)} - \log d^{(k-1)} \|_\infty \leq 0.1$, and $\beta^{-1} A^T D^{(0)} A \preceq A^T D^k A \preceq \beta A^T D^{(0)} A$ holds for $\beta = poly(m)$.*

The intuitive reasoning for the approach is that if we are able to show that the leverage scores are stable, then we are able to argue that we have a upper bound for low-rank updates. Lee and Sidford then show that assuming σ-stability, we also have stability for vectors along the leverage score norm:

Lemma 12. *For all $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and any vectors $\vec{x}$, $\vec{y} \in \mathbb{R}^m$ such that $\| \ln \vec{x} - \ln \vec{y} \|_\infty$, we have $\| \ln \vec{\sigma}_A(\vec{x}) - \ln \vec{\sigma}_A(\vec{y}) \|_{\vec{\sigma}_A(\vec{x})} \leq e^\epsilon \| \ln \vec{x} - \ln \vec{y} \|_{\vec{\sigma}_A(\vec{x})}$.*

Let $\vec{\theta}_t \stackrel{\text{def}}{=} \exp(\ln x + t(\ln y - \ln x))$ denote a straight line from $\ln \vec{x}$ to $\ln \vec{y}$. We then have that $\| \vec{z} \|_{\sigma A(\vec{\theta}_t)} \leq e^\epsilon \| \vec{z} \|_{\sigma_A}$. Using Jensen's inequality, Lee and Sidford show

$$\| \ln \vec{\sigma}_A(\vec{x}) - \ln \vec{\sigma}_A(\vec{y}) \|_{\vec{\sigma}_A(\vec{x})} \leq e^\epsilon \cdot \int_0^1 \| \frac{d}{dt} \ln \vec{\sigma}_A(\vec{\theta}_t) \|_{\vec{\sigma}_A(\vec{\theta}_t)} \, dt$$

Let $\mathbf{J}_{\ln \vec{z}}(\ln \vec{\sigma}_A(\vec{z}))$ denote the Jacobian matrix such that each cell i, j is $[\frac{\partial \ln_{\vec{\sigma}}(\vec{z})_i}{\partial \ln \vec{z}_j}]_{ij} = \frac{\partial \ln_{\vec{\sigma}}(\vec{z})_i}{\partial \vec{z}_j} z_j]_{ij}$. Lee and Sidford prove that the following holds for all vectors z and u:

$$\| \mathbf{J}_{\ln \vec{z}}(\ln \vec{\sigma}_A(\vec{z})) \vec{u} \|_{\vec{\sigma} A} \leq \| \vec{u} \|_{\vec{\sigma} A}$$

Using this fact, we can prove the lemma as follows:

$$\begin{aligned}\| \ln \vec{\sigma}_A(\vec{x}) - \ln \vec{\sigma}_A(\vec{y}) \|_{\vec{\sigma}_A(\vec{x})} &\leq e^{\epsilon} \cdot \int_0^1 \| \frac{d}{dt} \ln \vec{\sigma}_A(\vec{\theta_t}) \|_{\vec{\sigma}_A(\vec{\theta}_t)} \, dt \\ &= e^{\epsilon} \cdot \int_0^1 \| J_{\ln(\vec{\theta_t})}(\ln \vec{\sigma}_A(\vec{\theta_t}))(\frac{d}{dt} \ln(\vec{\theta_t})) \|_{\vec{\sigma}_A(\vec{\theta}_t)} \\ &\leq e^{\epsilon} \| \ln \vec{x} - \ln \vec{y} \|_{\vec{\sigma}_A(\vec{x})}\end{aligned}$$

Lee and Sidford then show how we can use leverage scores to maintain the inverse of $A^T DA$. They rely on two key results regarding leverage scores, proved in previous papers. The first states that one can sample $\widetilde{O}(n)$ rows of A according to their leverage score and obtain a spectral approximation $A^T HA$. The second states that given a solver for $A^T A$, one can efficiently compute approximate leverage scores σ for A.

Let $d \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$ be defined as before. Let $\sigma^{(apr)}$ be a vector of approximate leverage scores. Let H be a diagonal matrix maintained at each iteration. Lee and Sidford's algorithm under the σ-stability is similar to the algorithm under ℓ_2 stability, except for a few key differences:

- On input d, create a new leverage score vector $\sigma^{(apr)}$ such that $0.99\sigma_i \leq \sigma(d^{(k)})_i \leq 1.01\sigma_i$.
- For each i, check if either $0.9\sigma_i^{(old)} \leq \sigma_i^{(apr)} \leq 1.1\sigma_i^{(old)}$ or $0.9d_i^{(old)} \leq d_i^{(k)} \leq 1.1d_i^{(old)}$.
- If this constraint is violated, set:

  $$d_i^{(old)} = d_i^{(k)}$$

  $$\sigma_i^{(old)} = \sigma_i^{(apr)}$$

  Then set $h_i^{(k)} = \frac{d_i^{(k)}}{\min\{1, \gamma \cdot \sigma_i^{(apr)}\}}$ with probability $\min\{1, \gamma \cdot \sigma_i^{(apr)}\}$, and zero otherwise. Note that γ is a parameter defined from the leverage score results, which is used to maintain H.
- If the constraint is not satisfied, set $h_i^{(k)} = h_i^{(k-1)}$.
- Return the approximate inverse $(A^T H^{(k)} A)^{-1}$ at each iteration using Algorithm 4.

4.4 Complexity

We first show that the number of coordinate changes in $H^{(k)}$ over all k rounds is $O(r^2 \log n)$. For the ith row in H to be re-sampled, either $\sigma_A(d)_i$ or d_i has changed by more than a multiplicative constant. These events happen with probability $O(\gamma \sigma_A(d)_i)$ assuming independence of d and σ. Because $\gamma \stackrel{\text{def}}{=} 1000 c_s \log n$ for some constant c_s, the probability of sampling row i changing the matrix $A^T HA$ is $O(\sigma_A(d)_i \log n)$ by union bound.

From the σ-stability assumption, we know that $\sigma_A(d)_i$ does not change much each iteration. Similarly, from Lemma 12, we know that this implies that d_i also does not change much each iteration. In particular, we have the following inequalities:

$$\sum_k \| \ln d^{(k+1)} - \ln d^{(k)} \|_{\sigma_A(d^{(k)})} \leq O(r)$$

$$\sum_k \| \ln \sigma_A(d^{(k+1)}) - \ln \sigma_A(d^{(k)}) \|_{\sigma_A(d^{(k)})} \leq O(r)$$

Furthermore, re-sampling the ith row indicates that the sum of the σ norm square of either $\ln \sigma_A$ or $\ln d$ is at least $\sigma_A(d)_i/r$, and with probability $O(\sigma_A(d)_i \log n)$, the row is changed. Thus, the total number of rows changed over r iterations is $O(r^2 \log n)$.

Lee and Sidford then use this to prove the following theorem:

Theorem 13. *Suppose that the inverse maintenance problem satisfies the σ stability assumption. Then there is an algorithm that maintains a $\widetilde{O}(nnz(A) + n^2)$-time solver with high probability in time $\widetilde{O}(n^\omega + r(nnz(A) + n^2))$, where r is the number of rounds.*

Each iteration, we have that $D^{(old)} \approx_{0.2} D^{(k)}$ and all $\Sigma^{(old)} \approx_{0.2} \Sigma(d^{(k)})$, where $\Sigma \stackrel{\text{def}}{=} diag(\sigma)$. Using a well-known theorem on leverage scores, this implies that $A^T H^{(k)} A \approx_{0.1} A^T D^{(k)} A$, and thus the algorithm returns the correct result.

In terms of run-time, there are $\widetilde{O}(r^2)$ coordinate changes during the algorithm on expectation. Letting $\alpha = \widetilde{O}(r^2)$ and $s = \widetilde{O}(n)$, we can apply Theorem 6. Thus, the average cost to maintain $(A^T H^{(k)} A)^{-1}$ is $\widetilde{O}(\frac{n^\omega}{r} + nr^\omega + r^{2\omega})$. If we restart the algorithm every $r = (mn^{\omega-1})^{\frac{1}{2\omega+1}}$, Lee and Sidford then show we can achieve a time of $\widetilde{O}(n^\omega + rn^{\frac{2\omega^2}{2\omega+1}}) = \widetilde{O}(n^\omega + rn^2)$. Taking into account the time of computing the leverage scores at each iteration, the total time becomes

$$\widetilde{O}(n^\omega + r(\text{nnz}(A) + n^2))$$

Finally, note that this proof also gives us an LP solver in time $\widetilde{O}((\text{nnz}(A)+n^2)\sqrt{n}\log(1/\epsilon))$. They cite a previous paper that can solve an LP in $r = \sqrt{n}\log(1/\epsilon)$ rounds of inverse maintenance under the σ-stability assumption. Using the fact that their algorithm requires $\widetilde{O}(n^\omega + \text{nnz}(A))$ preprocessing time $\widetilde{O}(\text{nnz}(A) + n^2)$ time per iteration to maintain an inverse under the σ-stability assumption. Letting L be the bit complexity as defined before, their LP solver takes time:

$$\widetilde{O}((\text{nnz}(A) + n^2)\sqrt{n}L)$$

5 Conclusion

Solving linear programs is central to many of the most important optimization algorithms. These include algorithms for max-flow problems, constrained optimization problems, and many other problems that include a positive definite linear system. Since 1984, interior point methods have been the most efficient way of solving such optimization problems in polynomial time.

Our survey covered the history of interior point methods, focusing on methods for speeding up the cost of each iteration. In particular, we discussed various methods that have been used for improving the run-time cost of maintaining and updating a Hessian matrix at each iteration.

In Ëfficient inverse maintenance and faster algorithms for linear programming¨; Lee and Sidford employ many of the ideas we have seen before in approximating this inverse, such as

the use of faster linear algebra and low-rank update formulas. In addition, Lee and Sidford also introduce the novel idea of using subspace embeddings and leverage scores to provide and even cruder approximation to the Hessian inverse. Ultimately, their algorithm improves the fastest known LP-solver, and is near-optimal for sparse matrices.

A Linear Programming

A.1 Overview of Linear Programming

Linear programs helps us define problems where we want to optimize a linear objective function subject to constraints. Linear programs have two common canonical forms: the *primal* form and *dual* form.

Definition 14. *Linear programs can be expressed in the following primal (standard) form:*

- *minimize* $c^T x$
- *subject to* $Ax \leq b$
- $x \geq 0$

where $c, x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$ which is assumed to have full rank. It is also known that linear programs also have an equivalent dual form:

- maximize $b^T y$
- subject to $A^T y \geq c$
- $y \geq 0$

where $y \in \mathbb{R}^m$. Let x^* be the optimal solution to the primal program and y^* be the optimal solution to the dual program. The following equality holds for all linear programs:

$$c^T x^* = b^T y^*$$

and is referred to as the Strong Duality property. We also specify the definition of a *polytope*, which geometrically defines the feasible region of a linear program (x solution vectors that satisfy all constraints):

Definition 15. *Let* $P = \{x | Ax = b, x \geq 0\}$. *If* $P \neq \emptyset$ *and* P *is bounded, then* P *is a polytope.*

A linear programming problem can either 1) have a finite solution set, 2) not have a solution (i.e. have contradicting constraints such that P becomes $\emptyset$), or 3) have infinitely many solutions (i.e. have a solution set that is unbounded). Most Linear Program solvers assume that P is bounded and with at least one solution. Note that this implies that $m > n$, but in many practical applications m and n are of the same order.

It is well known that linear programs do not have a solution if and only if $\exists y$ s.t. $y^T A = 0, y^T b \neq 0$. Additionally, for any x to be an optimal solution, it must the case that x is a vertex.

Definition 16. *A point* $x \in P$ *is a vertex iff it is not a convex combination of other points in P, namely:*

$$\exists y^1, \ldots, y^{n+1} \in P$$

$$\alpha_1, \ldots, \alpha_{n+1} \geq 0 \text{ s.t. } x = \sum_{i=1}^{n+1} \alpha_i y^i, \sum_{i=1}^{n+1} \alpha_i = 1, \forall i \in [n+1], y^i \neq x$$

Specifically, we care about the linear programming solutions that are feasible and optimal. x is a feasible solution if it satisfies all constraints. x is optimal if it satisfies all constraints and there is no better solution for the objective.

B Solving LPs Prior to Interior Point Methods

The naive algorithm for linear problems is to do an exhaustive search of the feasible region, and to find the one that is optimal. Obviously, this algorithm is not efficient, as the feasible region is potentially unbounded. The objective in earlier studies of linear programming was to find more efficient algorithms with improved iterations through the constraints. The *Simplex Method* and the *Ellipsoid Method* are two methods prior to Interior Point Methods that were extensively used.

B.1 Simplex Method

Algorithm 3 Simplex Method

Initialize step $t = 0$
Select any vertex that is defined by n tight, linearly-independent constraints, x^0
while a neighbor that further minimizes $c \cdot x$ exists **do**
    Increase t by 1
    Select any neighbor $y \in N(x^{y-1})$ with direction $-c$, s.t. $c \cdot y < c \cdot x^{t-1}$
    Input $x^t = y$
end while
return x^t

Although there have been studies that suggest that the performance of the Simplex method is close to polynomial, it is exponential in the worst case scenario.

B.2 Ellipsoid Method

In order to improve the algorithmic run time of the Simplex Method, in 1979, Khachiyan provided the Ellipsoid Method that was the first algorithm to be weakly polynomial in run time.

Definition 17. *A general ellipsoid is E(a) =* $\{y \in \mathbb{R}^n : (y-a)^T A^T A(y-a) \leq r^2\}$*, where A is a full-rank n by n matrix. Note that if we would like to specifically talk about* $r^2 = 1$*, we can normalize our equation by readjusting a.*

Algorithm 4 Ellipsoid Method

Start with $k = 0$, $E_0 = E(a_0, A_o) \supseteq P$, P = $\{x : Cx \leq d\}$
while $a_k \notin P$ **do**
    Let $c^T x \leq d$ be an inequality that is valid for all $x \in P$ but $c^T a_k > d$.
    Let b = $\dfrac{A_k c}{\sqrt{c^T A_k c}}$
    Let $a_{k+1} = a_k - \frac{1}{n+1} b$
    Let $A_{k+1} = \dfrac{n^2}{n^2-1}(A_k - \dfrac{2}{n+1} bb^T$
end while
return x^t

Note that this algorithm is a reduction in which we assume that we can find an answer to whether the algorithm is feasible (i.e. decide if polytope is empty), and can in practice use binary search algorithms to guess efficiently. The intuition to select any vertex that is defined to be feasible, check if it is indeed feasible, and readjust our guess and continue. If the starting ellipsoid E_0 has volume V, then in $O(n \log \frac{V}{\epsilon})$ steps, E_t will have at least a volume of ϵ.

C Lee and Sidford's Path Finding Method

C.1 Overview

Lee and Sidford's [7] path finding paper extends the approach from the interior point method to match the polylogarithmic limit by iterating $\widetilde{O}(\sqrt{\text{rank}(A)}L)$ times. Their algorithm achieves a nearly linear run time with respect to $\text{nnz}(A)$ for fixed n value which is a significant improvement from existing algorithms. Their paper uses a technique called weighted path finding, which is central to many interior point methods.

They note that the log-barrier function varies greatly as the constraints of the linear program do, and that in particular, repeated constraints will slow down the path finding convergence. Therefore, the paper modifies the weights of this logarithmic barrier during the run-time of the algorithm, introducing a weight function g which maps slack values to a weight, in attempt to converge at a faster rate.

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

With this function, they show that the convergence rate depends on the dimension of the polytope, $\text{rank}(A)$, instead of the number of constraints. Furthermore, instead of relying solely on the current point x to compute the weights, they maintain a weight vector w and current point x separately in using the following barrier function:

$$\phi(w, x) = -\sum_{i \in [m]} w_i \log s(x)_i$$

The path finding algorithm works by maintaining two invariants:

- x is centered
- w multiplicatively close to $g(x)$.

Roughly speaking, the paper shows that through certain choices of updating, t can increase greatly as x remains close to the weighted central path, and $\|w\|_1$ remains small, allowing them to solve $\widetilde{O}(1)$ linear systems in each iteration, of which there are a total of $\widetilde{O}(\sqrt{\text{rank}(A)}L)$.

C.2 The Weighted Path

Let $w \in \mathbb{R}^m_{>0}$ be a set of positive weights in at each iteration of the LP solver. Let S^0 be the feasible region such that $S^0 \stackrel{\text{def}}{=} \{x \in \mathbb{R}^n : Ax > b\}$. At each iteration, Lee and Sidford

attempt to minimize the penalized objective function $f_t : S^0 \times \mathbb{R}^m_{>0} \to \mathbb{R}$, for all x and w with the following objective:

$$f_t(x, w) \stackrel{\text{def}}{=} tc^T x - \sum_{i \in [m]} w_i \log s(x)_i \tag{1}$$

where $s(x)_i$ is the slack for constraint i. Note that any point $x \in S^0$ can be expressed as a minimum of $f_t(x, w)$ given some choice of weights w. However, the solution to the LP is the $\{x, w\} \in \{S^0 \times \mathbb{R}^m_{>0}\}$ where t is large and $||w||_1$ is small.

Lee and Sidford measure the *centrality* of a point by calculating the size of the Newton step with respect to the Hessian norm at x. The centrality of a point tells us how close the point is to the central path. Note that $\nabla^2 f(x) = A^T S^{-2} A$ where $S \stackrel{\text{def}}{=} diag(s(x))$ is the Hessian for the non-weighted modified objective. With the weighted objective, the Hessian matrix becomes $AS^{-1}WS^{-1}A$ where $W \stackrel{\text{def}}{=} diag(w_i)$. The centrality of a (x, w) pair is then defined as the following:

$$\delta_t(x, w) = \| \ h_t(x, w) \ \|_{\nabla^2_{x,x} f_t(x,w)} = \| \ tc - A^T S^{-1} w \ \|_{(A^T S^{-1} W S^{-1} A)^{-1}} \tag{2}$$

where $h_t(x, w)$ is the Newton step along the weighted Hessian. Note that a central path point satisfies $\delta_t(x, w) = 0$. Lee and Sidford then bound the centrality of a weighted path step:

$$\delta_{(1+\alpha)t}(x, w) \leq (1 + \alpha)\delta_t(x, w) + \alpha\sqrt{||w||_1} \tag{3}$$

However, note that we also want to say that each Newton step can decrease $\delta_t(x, w)$, and center our path. To do so, Lee and Sidford bound how much the Hessian $A^T S^{-1} W S^{-1} A$ can change as we change x. Let $x^{(new)} = x + \Delta$, and let $s^{(new)}$ and s denote the slack associated with $x^{(new)}$ and x respectively. Lee and Sidford show that choosing $\Delta = -h_t(x, w)$ yields

$$||S^{-1} A \Delta||_\infty \leq \delta_t(s, w) \max_{i \in [m]} \| \ W^{-1/2} \mathbb{1}_i \ \|_{P_{S^{-1}A}(w)} \tag{4}$$

where $P_{S^{-1}A}(w)$ is the projection of w on the matrix $S^{-1}A$. This implies that as $\| \ W^{-1/2} \mathbb{1}_i \ \|_{P_{S^{-1}A}(w)}$ decreases, the stability of the Hessian increases. Lee and Sidford define $\gamma(s, w) \stackrel{\text{def}}{=} \| \ W^{-1/2} \mathbb{1}_i \ \|_{P_{S^{-1}A}(w)}$ as the *slack sensitivity* during a Newton step. The slack sensitivity relates how much slack can change during a given Newton step. The goal is to keep $\| \ w \ \|_1$ small while keeping centrality and $\gamma(s(x), w)$ small.

C.3 Centering

To ensure that $\delta_t(x, w)$ decreases, we can either perform a Newton step on x (move x along the central path) or set w such that $\delta_t(x, w) = 0$ (move the path closer to x). Instead of performing either step, Lee and Sidford use a centering step referred to as the r-step to do both, while maintaining the Newton step guarantee. In particular, r controls how x and w are changed such that $r = 0$ corresponds with a Newton step and $r = \infty$ corresponds to changing w to make x centered.

Definition 18. *Given a feasible point $\{x^{(old)}, w^{(old)}\} \in \{S^0 \times \mathbb{R}^m_{>0}\}$ and a path parameter t, the r-step, $\{x^{(new)}, w^{(new)}\} = step_t(x^{(old)}, \vec{w}^{(old)}, r)$ is defined as:*

$$x^{(new)} = x^{(old)} - \frac{1}{1+r}\vec{h}_t(x^{(old)}, w^{(old)})$$
$$w^{(new)} = w^{(old)} + \frac{r}{1+r}W_{(old)}S^{-1}_{(old)}Ah_t(x^{(old)}, w^{(old)})$$

and $h_t(x, w)$ is the Newton step.

We note that the r-step performs a multiplicative update on the weights that is r times larger than the update on the slacks. Following defining the r-step, Lee and Sidford show that the r-step is stable, and does not change the Hessian matrix too much at each iteration:

$$(1-\delta_t\gamma)\nabla^2_{xx}f_t(x^{(old)}, w^{(old)}) \preceq \nabla^2_{xx}f_t(x^{(new)}, w^{(new)}) \preceq (1+\delta_t\gamma)\nabla^2_{xx}f_t(x^{(old)}, w^{(old)}) \quad (5)$$

where γ is the slack sensitivity on $(x^{(old)}, w^{(old)})$ and $\delta_t = \delta_t(x^{(old)}, w^{(old)}) \leq \frac{1}{8\gamma}$. Additionally, under these assumptions, Lee and Sidford show that the centrality of each r-step can be bounded based on the previous iteration δ_t:

$$\delta_t(x^{(new)}, w^{(new)}) \leq \frac{2}{1+r}\gamma\delta_t^2 \quad (6)$$

This is their first bound on the change in centrality $\delta_t(x^{(new)}, w^{(new)})$ at each iteration.

C.4 Weight Functions

In order to show that this algorithm can converge in $\widetilde{O}(\sqrt{\text{rank}(A)}L)$ iterations, Lee and Sidford use a weight function $g : \mathbb{R}^m_{>0} \to \mathbb{R}^m_{>0}$ from slack values to positive weights. We also define $G(s) \stackrel{\text{def}}{=} diag(g(s))$. The constraints to the weight function constraints are the following:

- Size: The weight function yields small size $||g(s)||_1 \leq c_1(g)$.
- Slack Sensitivity: The weight function does not change too much as the slacks change: $\gamma(s, g(s)), 1 \leq c_\gamma(g)$.
- Step Consistency: g does not change too much as x changes at each r-step: $||(I + r^{-1}G(s)^{-1}G(s)S)y||_\infty, 1 \leq ||y||_\infty + c_r||y||_{G(s)}$.
- Uniformity: $||g(s)||_\infty \leq 2$.

We use constants c_1, c_γ, and c_r to define these bounds. Furthermore, because $g(s)$ may be expensive to compute, Lee and Sidford note that we will need to store the weights separately and define $\Psi(s, w) \stackrel{\text{def}}{=} \log(g(s)) - \log(w)$ as the error in computing the weight.

Rather than computing $g(s)$ at each step, they maintain w separately and ensure that $||\Psi(s, w)||_\infty$ is small after each r-step. Lee and Sidford show that under certain conditions, each r-step does not increase $\Psi(s, w)$ by too much. In particular, if δ_t and $||\Psi(s^{(old)}, w^{(old)})||_\infty$ are small enough, then the following is satisfied for $\Delta = \Psi(s^{(new)}, w^{(new)}) - \Psi(s^{(old)}, w^{(old)})$:

$$||\Delta||_\infty \leq 4c_\gamma\delta_t \quad (7)$$

$$||\Delta||_{W^{(new)}} \leq \frac{e^\epsilon c_r}{1+c_r}\delta_t + 13c_\gamma\delta_t^2 \quad (8)$$

At this point, Lee and Sidford assume we have exact weights and a computable function $g(s)$. They bound the rate of convergence of the path following algorithm. Letting $\epsilon_\infty \stackrel{\text{def}}{=} ||\log(w^{(new)}) - \log(w^{(old)})||_\infty \leq \frac{1}{2}$. Lee and Sidford show the following inequality holds:

$$\delta_t(x, w^{(new)}) \leq (1 + \epsilon_\infty)[\delta_t(x, w^{(old)}) + ||\log(w^{(new)}) - \log(w^{(old)})||_{W^{(old)}}] \tag{9}$$

Now, assume we have an efficiently computable exact weight function $g(s)$. Let $\delta_t \stackrel{\text{def}}{=} \delta_t(x^{(old)}, g(s^{(old)})) \leq \frac{1}{80c_\gamma c_r}$. Lee and Sidford show that our centrality improves at each iteration!

Theorem 19. *For a weight function g, let $x^{(old)} \in S^0$ and $x^{(new)}$ be the result after applying centering step. If $\delta_t \stackrel{def}{=} \delta_t(x^{(old)}, g(s^{(old)})) \leq \frac{1}{80c_\gamma c_r}$, then the following inequality holds at each iteration:*

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

Proof. Note that since we assume $g(s)$ is computed exactly, we have that $\Delta = \log(\frac{g(s^{(new)})}{w^{(new)}})$. Additionally, we have that $g(s^{(new)}) = w^{(old)}$ and thus $\Delta_\infty = \epsilon_\infty$. The proof begins with equation 9:

$$\delta_t(x^{(new)}, g(s^{(new)})) \leq (1 + \epsilon_\infty)[\delta_t(x^{(new)}, w^{(old)}) + ||\log(w^{(new)}) - \log(w^{(old)})||_{w^{(old)}}]$$

Using equation 7:

$$\leq (1 + 4c_\gamma\delta_t)[\delta_t(x^{(new)}, w^{(old)}) + ||\log(w^{(new)}) - \log(w^{(old)})||_{w^{(old)}}]$$

Using equation 8:

$$\leq (1 + 4c_\gamma\delta_t)[\delta_t(x^{(new)}, w^{(old)}) + \frac{e^\epsilon c_r}{1 + c_r}\delta_t + 13c_\gamma\delta_t^2]$$

Using equation 6:

$$\leq (1 + 4c_\gamma\delta_t)[c_\gamma\delta^2 + \frac{e^\epsilon c_r}{1 + c_r}\delta_t + 13c_\gamma\delta_t^2]$$
$$\leq (1 - \frac{1}{4c_r}\delta_t)$$

□

At this point, we can see exactly how the algorithm converges to the central path. In particular, if $\delta_t(x, g(s))$ is $O(c_\gamma^{-1}c_r^{-1})$ it takes $O(c_r^{-1})$ iterations to decrease $\delta_t(x, g(s))$ by a multiplicative constant. Additionally, from equation 3, we can increase t by a multiplicative $(1+O(c_\gamma^{-1}c_r^{-1}c_1^{-1/2}))$ and maintain $\delta_t(x, g(s)) = O(c_\gamma^{-1}c_r^{-1})$ using $O(c_\gamma^{-1}c_r^{-1}c_1^{-1/2})$ iterations of exact centering. Of course, this only holds if we compute the weight function $g(s)$ exactly, which will not be the case in the actual algorithm.

At this point, Lee and Sidford choose a weight function that will allow a $\widetilde{O}(\sqrt{\text{rank}(A)}L)$ convergence rate:

$$g(s) \stackrel{\text{def}}{=} \arg\min_w \hat{f}(s, w) \tag{10}$$

$$\hat{f}(s,w) = \mathbb{1}^T w - \frac{1}{\alpha}\log\det(A_s^T W^\alpha A_s) - \beta\sum_{i\in m}\log w_i \tag{11}$$

where α and β are parameters. This leads to the following theorem (we omit a description of the proof for conciseness):

Theorem 20. *Define α and β as the following:*

$$\alpha = 1 - \frac{1}{\log(\frac{2m}{rank(A)})} \text{ and } \beta = \frac{rank(A)}{2m}$$

Then the following parameters for g hold:

- *Size:* $c_1(g) = 2rank(A)$
- *Slack sensitivity:* $c_\gamma(g) = 2$
- *Step consistency:* $c_r(g) = 2\log(\frac{2m}{rank(A)})$

In order to carry out the LP solver, Lee and Sidford approximate $g(s)$ at each step using gradient descent method. Because the weight function is convex, Lee and Sidford show that $g(s)$ can be computed to high accuracy in $\widetilde{O}(1)$ iterations assuming we start with a weight w not too far from the optimum and we compute the gradient of $\hat{f}$ exactly.

While the first constraint is not an issue as we ensure g does not change too much between iterations, computing the gradient of $\hat{f}$ is problematic due to the presence of a projection matrix $diag(P_{S^{-1}A}(x)) \stackrel{\text{def}}{=} \sigma_{S^{-1}A}(x)$. However an efficient approximation can be done using the JL-Lemma. In particular, Lee and Sidford note that the leverage scores of the projection $[\sigma_{S^{-1}A}]_i$ exactly correspond with the ℓ_2 length of $P_A(x)\mathbb{1}_i$. Thus, they can use the JL-lemma to approximate the leverage scores of each constraint $[\sigma_{S^{-1}A}]_i$ with a projection onto a low-dimensional subspace. This means the algorithm only needs to solve $\widetilde{O}(1)$ regression problems in order to approximate $\sigma_{S^{-1}A}$. This can be done in $O(\epsilon^{-2}\log m)$ time.

To approximate the initial weights without having an approximate weight to help the computation, Lee and Sidford use an iterative function that requires $\widetilde{O}(\sqrt{\text{rank}(A)})$ iterations of the computeWeight function, decreasing a penalty β term gradually. The time for computing the initial weights is dominated by the time needed to solve $\widetilde{O}(\sqrt{\text{rank}(A)}\log(1/K)K^2)$ linear systems.

C.5 Modifications

Ultimately, if multiplicative approximation to the weight function is computed, Lee and Sidford show that the running-time still holds. The use of noisy oracle further decreases the run-time but requires maintaining the invariant $||\Phi(s,w)||_\infty \leq K$ for some error threshold K at each iteration (where $\Phi(s,w) = \log(g(s)) - \log(w)$ as before).

Maintaining a multiplicative approximation requires a slightly modified algorithm, balancing the centrality δ_t with moving w towards $g(s)$ between c_r-steps. At each step, the algorithm takes a step down the central path in order to decrease δ_t. At the same time, the algorithm also moves w toward $g(s)$ without increasing δ_t too much.

This modified algorithm requires smoothing out changes to the weights by using a slowly changing approximation to g. They achieve this smooth approximation by presenting a

solution to the *chasing 0 game*, a generalization of the smoothing problem. The solution to the chasing 0 game allows Lee and Sidford to produce an adversarial weighted path following strategy that has only a multiplicative approximation to the weight function. We omit the discussion and analysis of the chasing 0 game solution for conciseness.

The result is a modification of the centering algorithm known as *centeringInexact*, which requires additional analysis. The main idea is that smoothness ensures the weight cannot move too far away from $g(s)$ at each c_r-step. In this way, we can change w back without hurting centrality δ_t. With this strategy, Lee and Sidford show that each iteration the centrality improves at each iteration while maintaining a bound on the weight error $\Psi(x^{(new)}, w^{(apr)})$.

C.6 Algorithm and Complexity

Algorithm 5 pathFollowing$(x^{(old)}, w^{(old)}, t_{start}, t_{end})$

$c_r = 2\log(\frac{2m}{\text{rank}(A)})$, $t = t_{start}$, $K = \frac{1}{24c_r}$

while $t < t_{end}$ **do**

$(x^{(new)}, w^{(apr)}) = \text{centeringInexact}(x^{(old)}, w^{(old)}, K, \text{computeWeight})$

$t := t(1 + \frac{1}{10^{10}c_r^3 \log c_r m\sqrt{\text{rank}(A)}})$

$(x^{(old)}, w^{(old)}) = (x^{(new)}, w^{(apr)})$

Every $\frac{m}{100c_r \log(c_r m)}$ steps, check if x and w satisfy Ψ and δ invariants, if not then roll back to last time invariants were met.

end while

Output $(x^{(old)}, w^{(old)})$

We combine all subroutines in Algorithm 3. Note that computeWeight is the gradient descent algorithm used for approximately computing w and centeringInexact is the centering algorithm used for stepping forward while maintaining $w \approx g(s)$.

Using this algorithm, Lee and Sidford show that computing $(x^{(new)}, w^{(new)})$ for a given number of iterations specified by t_{end} and t_{start} takes $\widetilde{O}(\sqrt{\text{rank}(A)}\log\frac{t_{start}}{t_{end}})$ iterations, where each iteration requires solving an $\widetilde{O}(1)$ linear system. In addition, the algorithm ensures a decrease in centrality. With a few minor modifications (obtaining an initial point, rounding optimal point to a vertex, deal with unbounded solutions, etc.), Lee and Sidford are able to prove Theorem 21.

Theorem 21. *Let L denote the bit complexity of a linear program $\min_{x\in\mathbb{R}^n, Ax\geq b} c^T x$ and suppose for any positive definite matrix $D \in \mathbb{R}^{m\times m}$ there is an algorithm solve(A, b, D, ϵ) such that*

$$||solve(A, b, D, \epsilon) - (DA)^+ b||_{A^T D^2 A} \leq \epsilon ||(DA)^+ b||_{A^T D^2 A}$$

in time $O(\mathcal{T}\log(1/\epsilon))$ for any $\epsilon > 0$. with success probability greater than $1 - \frac{1}{m}$. Then, there is an algorithm to solve the linear program in expected time $\widetilde{O}(\sqrt{rank(A)}(\mathcal{T} + nnz(A))L)$ to find the active constraints of an optimal solution or prove that the program is unfeasible or unbounded. Using the optimal solve routine as the solve algorithm yields:

$$\widetilde{O}(\sqrt{rank(A)}(nnz(A) + rank(A)^\omega)L)$$

Lee and Sidford use the compute initial weight function which gives us an initial weighted central path point. It is possible that δ_t could be large for this point. Adding a modified cost function $c^{(new)} = A^T S^{-1} w$ we can get $\delta_t = 0$ for the new cost function. In particular, we can start with the cost function $t_{start}c$ and reach another cost function $t_{end}c$ in time that depends on $\log(t_{start}/t_{end})$. We can decrease t to get the central path point for $2^{-\Theta(L)}c^{(new)}$. Since this is close to zero, this means that δ_t is also close to zero, and we arrive at the central path.

Using Algorithm 3, we slowly bring the modified cost function $c^T x_t$ close to $c^T x$. This requires only $\widetilde{O}(L)$ iterations of centeringInexact, and thus the algorithm only does $\widetilde{O}(L)$ linear system solves. Using fast linear algebra and the current optimal solve routine, we get a total running time of:

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

With a few more modifications, centeringInexact can be computed in $\widetilde{O}(1)$ depth and $\widetilde{O}(m)$ work, yielding the following theorem:

Theorem 22. *There is an $\widetilde{O}(\sqrt{rank(A)}L)$ depth polynomial work algorithm to solve linear programs of the form:*

$$\min_{x \in \mathbb{R}^n, Ax \geq b} c^T x$$

where L denotes the bit complexity of the linear program

References

[1] Nisheeth K. Vishnoi. A mini-course on convex optimization: with a view toward designing fast algorithms, 2014.

[2] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 302–311, New York, NY, USA, 1984. ACM.

[3] Yu Nesterov and A Nemirovski. Acceleration and parallelization of the path-following interior point method for a linearly constrained convex quadratic problem. *SIAM Journal on Optimization*, 1(4):548–564, 1991.

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

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

[6] Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. *CoRR*, abs/1503.01752, 2015.

[7] Yin Tat Lee and Aaron Sidford. Matching the universal barrier without paying the costs : Solving linear programs with $\widetilde{O}(\sqrt{rank})$ linear system solves. *CoRR*, abs/1312.6677, 2013.