Complexity in continuous optimization (6/6)

March 6, 2025

Last lecture: Linear programming! (with nonlinear optimization)

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

arXiv > cs > arXiv:1910.08033

Search... Help

Computer Science > Data Structures and Algorithms

[Submitted on 17 Oct 2019 (v1), last revised 31 Aug 2020 (this version, v2)]

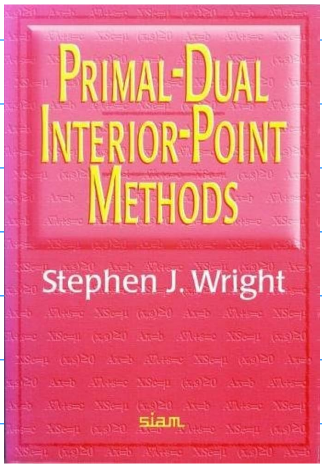
Solving Linear Programs with Sqrt(rank) Linear System Solves

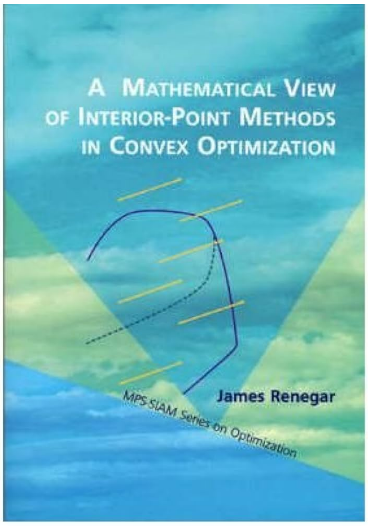
Yin Tat Lee, Aaron Sidford

We present an algorithm that given a linear program with n variables, m constraints, and constraint matrix A, computes an ϵ-approximate solution in $\tilde{O}(\sqrt{rank(A)}\log(1/\epsilon))$ iterations with high probability. Each iteration of our method consists of solving $\tilde{O}(1)$ linear systems and additional nearly linear time computation, improving by a factor of $\tilde{\Omega}((m/rank(A))^{1/2})$ over the previous fastest method with this iteration cost due to Renegar (1988). Further, we provide a deterministic polynomial time computable $\tilde{O}(rank(A))$-self-concordant barrier function for the polytope, resolving an open question of Nesterov and Nemirovski (1994) on the theory of "universal barriers" for interior point methods.

Applying our techniques to the linear program formulation of maximum flow yields an $\tilde{O}(|E|\sqrt{|V|}\log(U))$ time algorithm for solving the maximum flow problem on directed graphs with $|E|$ edges, $|V|$ vertices, and integer capacities of size at most U. This improves upon the previous fastest polynomial running time of $O(|E|\min\{|E|^{1/2},|V|^{2/3}\}\log(|V|^2/|E|)\log(U))$ achieved by Goldberg and Rao (1998). In the special case of solving dense directed unit capacity graphs our algorithm improves upon the previous fastest running times of $O(|E|\min\{|E|^{1/2},|V|^{2/3}\})$ achieved by Even and Tarjan (1975) and Karzanov (1973) and of $\tilde{O}(|E|^{10/7})$ achieved more recently by Mądry (2013).

Comments: The merged version of abs/1312.6677 and abs/1312.6713. It contains several new results beyond these prior submissions, including a nearly optimal self-concordant barrier and its relation to Lewis weight
Subjects: **Data Structures and Algorithms (cs.DS)**; Optimization and Control (math.OC)
Cite as: arXiv:1910.08033 **[cs.DS]**





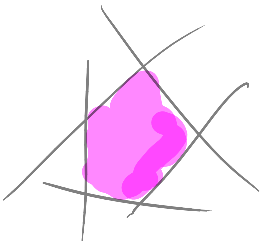
LINEAR PROGRAMMING AND COMPLEXITY

$$(P) \quad \underset{x \in \mathbb{R}^n}{\text{minimize}} \; c^T x \quad \text{s.t. } Ax \geq b$$

$\left(A = \begin{bmatrix} a_1^T \\ \vdots \\ a_\ell^T \end{bmatrix} \quad b \in \mathbb{R}^\ell \quad a_i^T x \geq b_i \;\forall i \right)$

$A \in \mathbb{R}^{\ell \times n}$, $\text{rank}(A) = n \leq \ell$ (typically $n \ll \ell$)

↳ Assume: Feasible set $\{x \in \mathbb{R}^n \mid Ax \geq b\}$ is a bounded polytope with $\{x \mid Ax > b\} \neq \emptyset$



Goal: Compute an approximate solution of (P) in polynomial time

→ Given a tolerance $\varepsilon \in (0,1)$, bound the number of iterations needed by an algorithm needed to reach an ε-approximate solution

+ cost of each iteration

→ In LP:
- The dependency on ε is not the main concern
- Key: Get lowest dependency on n and ℓ

(Weakly) Polynomial-time algorithm: Method for which the complexity is polynomial in n and ℓ

1 Simplex method

→ Dantzig 1940s

→ Philosophy: go from one vertex of the feasible set to another as long as you improve the objective

NB: The solution is always a vertex

→ A family of algorithms, defined essentially by the "pivoting rule" to move between vertices

⊕ Exact algorithm: It will find the solution in finitely many steps

⊖ Not a polynomial-time algorithm

⇒ Family of examples (Klee-Minty, 1960s)

- $\forall$ dimension n, define a polytope with $l = 2n$ inequalities and 2^n vertices
- Show that a version of the simplex (a pivoting rule) and an initialization such that the method takes 2^n iterations to get the optimum (visits all vertices)

⚠ Worst-case behavior: The simplex method works much better in practice

(Aside: HiGHS : Open-source solver for LPs based on the simplex method)

② Ellipsoid method

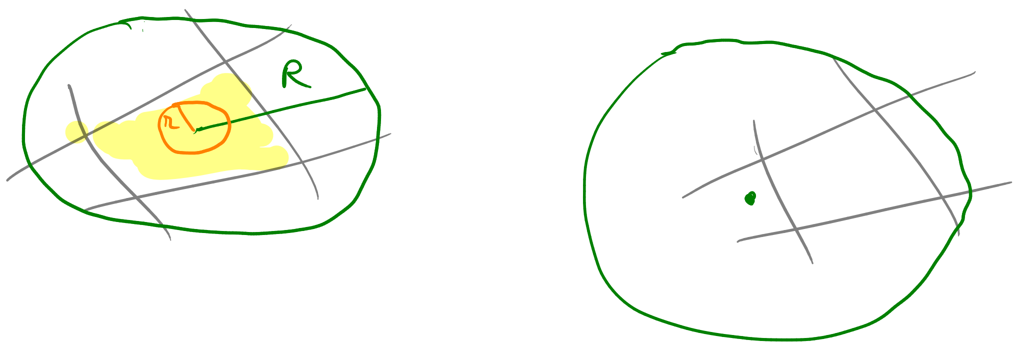
Kachiyan 1979 : Algorithm with a complexity polynomial in n and ℓ

Result : Find x_ε such that $c^T x_\varepsilon \leq c^T x^* + \varepsilon$

(where $x^* \in \operatorname{argmin} c^T x$ s.t. $Ax \geq b$) in at most

$$\underbrace{O(1)}_{\text{universal constant}} \times \ell^2 \times \ln\left(2 + \frac{R}{r} \times \frac{1}{\varepsilon}\right) \text{ iterations}$$

where $R > r > 0$ are such that $\{x \mid Ax \geq b\}$ is contained in a ball of radius R and contains a ball of radius r.



Algorithm : Ellipsoid method

Def: An ellipsoid in $\mathbb{R}^n$ has the form $\{z \mid z = Bu + c, \|u\| \leq 1\}$

$B \in \mathbb{R}^{n \times n}$, $B = B^T \succ 0$

$c \in \mathbb{R}^n$ center of the ellipsoid

Ball of radius ρ: $B = \rho I$

→ the method builds ellipsoids of decreasing volumes that all contain the solution

⇒ Stop whenever the volume of the ellipsoid is small enough

Initialization : Set E_0 to the ball of radius R that contains the ellipsoid ⇒ x_0 : center of the ellipsoid

Iteration k : Given $E_k = \{ y \mid B_k u + x_k, \|u\| \leq 1 \}$,

① call a separation oracle on x_k

Outcome 1 : the oracle says that $x_k \in \{x \mid Ax \geq b\}$

In that case, set $v_k = c$

Outcome 2 : The oracle says that x_k is infeasible, and returns a vector v_k such that

$$v_k^T (x_k - x) > 0 \quad \forall x \in \{x \mid Ax \geq b\}$$

② Define $\hat{E}_{k+1} = \{ x \in E_k : v_k^T x \leq v_k^T x_k \}$

Outcome 1 : Improve objective

—— 2 : —— feasibility

and build an ellipsoid E_{k+1} that contains $\hat{E}_{k+1}$ such that the volume of E_{k+1} is smaller than that of E_k (always possible)

(3) Stop if volume$(\mathcal{E}_{k+1}) < (r\varepsilon)^n$, otherwise increase k and go back to (1).

Key: Separation oracle → easy to define for polytopes
→ more generally, can be defined (and computed efficiently!) for bounded convex sets.

Harder part: compute $\mathcal{E}_{k+1}$: does not work that well in practice

(+) Complexity polynomial in n and l
(−) Not efficient in practice (and actually hard to implement)

→ From the theoretical side, people have extended ellipsoid methods to

a) The convex setting

$$\underset{x \in \mathbb{R}^n}{\text{minimize}}\ f(x) \quad \text{s.t.} \quad x \in \mathcal{X}$$

$f : \mathbb{R}^n \to \mathbb{R}$ convex (C^1) function
$\mathcal{X} \subseteq \mathbb{R}^n$ bounded convex set

Includes classes of QPs, SDPs, ...

Provided you can define a separating oracle on $\mathcal{X}$, you can implement the ellipsoid method!

$\Rightarrow$ change $v_k = c$ to $v_k = \nabla f(x_k)$

b) To Nonconvex optimization

(Hinder 2018) $\Rightarrow$ Ellipsoid technique for nonconvex optimization

"Convex until proven guilty"

$\rightarrow$ Gets the best known $O(\varepsilon^{-7/4})$ bound to satisfy $\|\nabla f(x_k)\| \leq \varepsilon$

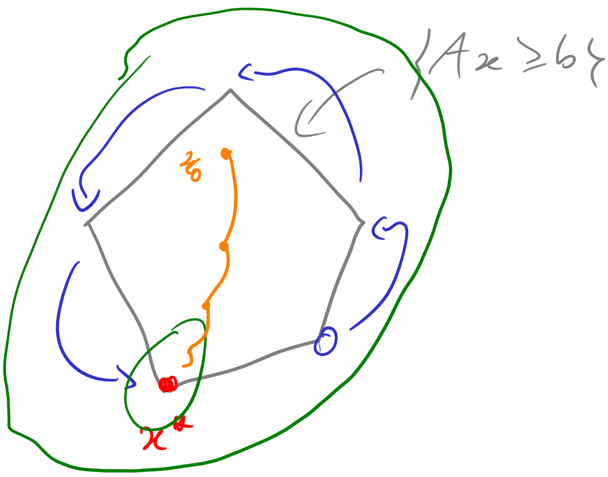
$\rightarrow$ From the computational side (and partly from a theoretical perspective), ellipsoid methods have been overperformed by interior-point methods

③ IPMs (Interior-Point Methods)

Breakthrough: Karmarkar (1984)

- Algorithm ≠ Ellipsoid with complexity
- Good practical performance (better than simplex)

$\Rightarrow$ IPM golden age: 1985 – 2000s, lead to the development of solvers like CPLEX, Gurobi



Simplex method: Go from vertex to vertex

$\longrightarrow$ Exact solution

Ellipsoid method: Build increasingly small ellipsoids

IPM: Start from a strictly feasible point $Ax > b$ and move in the interior of the feasible region

⊖ Never get the exact solution

⊕ Can get (ε-) close to the solution very quickly

$\longrightarrow$ Several families of IPMs: Primal-dual, dual path-following, barrier

$\longrightarrow$ Works for LPs but also for QPs, SDPs, SOCPs, ...

$\Rightarrow$ Many classes of convex programming

4 Complexity of barrier IPMs

$\longrightarrow$ Class of IPMs with best guarantees

$\longrightarrow$ Not the best class in practice in general (Primal-dual is!)

<u>Pb</u>: $\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad c^T x \quad \text{s.t.} \quad Ax \geq b$ $\qquad A \in \mathbb{R}^{\ell \times n}$

$(\text{rank}(A) = n \leq \ell)$

Barrier IPMs consider a sequence of problems of the form $(P_\mu) \quad \underset{x \in \mathbb{R}^n}{\text{minimize}} \; f_\mu(x) \quad$ for $\mu \geq 0$

$$f_\mu(x) = \mu \, c^T x + \beta(x)$$

linear objective

Barrier term defined according to $\{x \mid Ax \geq b\}$

β is chosen so that if x belongs to the boundary of the polytope, then $\beta(x) = +\infty$ (typically β not defined or ∞ outside of the polytope)

Also want $\beta(x) \longrightarrow \infty$
$\qquad x \longrightarrow \partial(Ax \geq b)$

<u>Examples</u>: • Log(arithmic)-barrier

$$\beta(x) = -\sum_{i=1}^{\ell} \ln(a_i^T x - b_i)$$

If $Ax > b$, $\ln$ well defined

If $\exists i, \; a_i^T x = b_i, \quad \beta(x) = +\infty$

$\Rightarrow$ Penalizes points that are close to the boundary

- Weighted log barrier

$$\beta(x) = -\sum_{i=1}^{\ell} w_i \ln(a_i^T x - b_i)$$ for some vector $w \in \mathbb{R}^\ell$

→ For a well-chosen barrier, (P_μ) is equivalent to the original LP for μ sufficiently large

→ Barrier IPMs solve a sequence (P_{μ_k}) with increasing μ_k

Algorithm

- Define $x_0 = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \; f_0(x) = \beta(x)$ → Point in the polytope that minimizes the proximity to the boundary

- Set $\mu_0 > 0$.

- For $k = 0, \ldots$

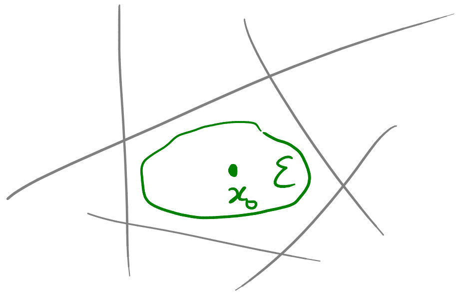
  Compute $x_{k+1} \in \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \; f_{\mu_k}(x)$.

  → Can do that numerically starting from x_k

  → Can also do it inexactly

  Pick $\mu_{k+1} > \mu_k$

Remark: $\underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \; \overbrace{\beta(x)}^{f_0(x)}$: "analytic center of the polytope with respect to β"



→ For a log-barrier, can define the Dikin ellipsoid

$$\mathcal{E} = \{x \mid (x-x_0)^T \nabla^2 \beta(x_0)(x-x_0) \leq 1\}$$

$$\Rightarrow \mathcal{E} \subseteq \{x \mid Ax \geq b\}$$

Complexity of that method (with exact minimization of f_{μ_k})

If $\mu_{k+1} = (1+\delta)\mu_k$ for some $\delta > 0$, then $\forall \varepsilon \in (0,1)$,

$\|x_{\mu_{k+1}} - x_{\mu_k}\| \leq \varepsilon$ after at most

$$\mathcal{O}\left(\delta^{-1} \ln(1/\varepsilon)\right) \text{ iterations}$$

⚠ If μ_k increases too quickly, minimizing f_{μ_k} becomes equivalent to solving the original problem!

Key: "Stay in the interior" $\Leftrightarrow$ μ_k should not increase too rapidly

ℓ : number of constraints

↳ Typical choices for δ are $\frac{1}{\ell}$ and $\frac{1}{\sqrt{\ell}}$. These choices are motivated by inexact solves of the subproblems using Newton's method

$$x_{k+1} \in \operatorname{argmin}_{x \in \mathbb{R}^n} f_{\mu_k}(x) \Longrightarrow \left\{ \begin{array}{l} \tilde{x}_0 = x_k \\ \text{For } j = 0 \ldots \\ \tilde{x}_{j+1} = \tilde{x}_j - \alpha_j \nabla^2 f_{\mu_k}(\tilde{x}_j)^{-1} \nabla f_{\mu_k}(\tilde{x}_j) \end{array} \right.$$

positive stepsize

Newton step (well defined for strongly convex barriers)

→ Analysis relies on the convergence of Newton's method for strongly convex functions

→ Barrier functions are chosen to be strongly convex and even self-concordant (has to do with $\nabla^3 f(\cdot)$)

→ The earlier results used just 1 iteration of Newton's method and small changes in μ_k (so that the convergence theory of Newton's method applies across all iterations)

who	# of iterations	iteration cost
Karmarkar (1984)	$\delta = \frac{1}{\ell}$ $O(\ell \ln(1/\varepsilon))$	1 linear system solve (invert $\nabla^2 f_{\mu_k}$)
Renegar (1986)	$\delta = \frac{1}{\sqrt{\ell}}$ $O(\sqrt{\ell} \ln(1/\varepsilon))$	1 linear system solve
Vaidya (1989)	$O(n \ln(1/\varepsilon))$ $(\ell >> n)$	n linear system solves ($\sim n$ its of Newton's method)
Nesterov Nemirovski (1994)	$O(\sqrt{n} \ln(1/\varepsilon))$	More expensive than the original problem

What they showed: there exists a barrier function so that barrier IPM has complexity $O(\sqrt{n} \ln(1/\varepsilon))$

BUT

- Minimizing f_{μ_k} with that barrier is more expensive than solving the LP
- Even evaluating the derivatives of the barrier is more expensive

$\simeq$ The perfect barrier for the problem, but too expensive to compute

Lee, Sidford (FOCS 2014)

Got the $O(\sqrt{m}\ \ln(1/\varepsilon))$ complexity with a per-iteration cost of $\tilde{O}(1)$ linear system solves

<u>Key idea</u>: Instead of using the "optimal" barrier, build a sequence of barrier functions to approximate that barrier

$\to$ Want to use a sequence of weights $\{w_k\}_k$ and barriers of the form $x \mapsto -\sum_{i=1}^{\ell}[w_k]_i \ln(a_i^T x - b_i)$

$$\underbrace{}_{f_{\mu_k}(x, w_k) = \mu_k c^T x - \sum_{i=1}^{\ell}[w_k]_i \ln(a_i^T x - b_i)}$$

<u>Algorithm</u>

Start with $\mu_0 > 0$, $x_0 \in \operatorname{argmin}_{x \in \mathbb{R}^n} f_0(x)$ (or simply $x_0 \in \{x \mid Ax > b\}$)

$w_0 \in \mathbb{R}^\ell$, $w_0 > 0$

Iteration k

① $x_{k+1} \simeq \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \; f_{\mu_k}(x, w_k)$] should not be too expensive

② Compute w_{k+1}] should be made to resemble the optimal barrier

③ Compute μ_{k+1} so that $\mu_{k+1} > \mu_k$] μ_k should not increase too quickly

Lee & Sidford showed that

③ can pick $\mu_{k+1} = (1+\delta)\mu_k$ with $\delta = O\left(\frac{1}{\sqrt{e}}\right)$

① $\tilde{O}(1)$ Newton steps
⇒ Guarantees on "Newton decrement"

At iteration k: $\nabla f_{\mu_k}(x_k, w_k)^T \, \nabla^2 f_{\mu_k}(x_k, w_k) \, \nabla f_{\mu_k}(x_k, w_k)$

Measures how far x_k is from the solution of $\underset{x}{\text{minimize}} \; f_{\mu_k}(x, w_k)$

Δ_k (≈ Lyapunov function)

$\forall k,\quad \Delta_{k+1} \le \left(1 - \frac{\gamma}{\sqrt{e}}\right) \Delta_k$ after $\tilde{O}(1)$ Newton steps

② Updating w_k

Key quantity: "slack vector" $s(x_k) = Ax_k - b$

$$w_{k+1} = \underset{\substack{w > 0 \\ w \in \mathbb{R}^d}}{\operatorname{argmin}} \; g_{k+1}(w)$$ → weights w_{k+1} are chosen according to $s(x_{k+1})$

$$g_{k+1}(w) = \sum_{i=1}^{\ell} w_i - \frac{1}{\alpha_1} \log\det\left(A^T S_{k+1}^{-T} W^{\alpha_1 - 1} S_{k+1}^{-1} A\right) - \alpha_2 \cdot \sum_{i=1}^{\ell} \log(w_i)$$

$$S_{k+1} = \mathrm{diag}(s(x_{k+1})) , \quad W = \mathrm{diag}(w)$$

$$\alpha_1 = 1 - \log_2\left(\frac{2\ell}{m}\right)^{-1} , \quad \alpha_2 = \frac{m}{2\ell}$$

Application : Max flow

- Graph $G = (V, E)$ $\quad \underbrace{|E|}_{\ell} \gg \underbrace{|V|}_{m}$

Max Flow

$$\underset{f \in \mathbb{R}^{|E|}}{\text{maximize}} \; c^T f \quad \text{s.t.} \quad B^T f = 0, \quad 0 \le f \le u$$

Graph community: Best complexity was

$$\tilde{O}\left(|E| \min\{|E|^{1/2}, |V|^{3/2}\}\right)$$

$$= \tilde{O}\left(|E|\,|V|^{3/2}\right) \text{ if } |E| \gg |V|$$

(Goldberg & Rao 1998)

IPMs (Lee & Sidford) $\quad \tilde{O}\left(|E|\,|V|^{1/2}\right)$

<u>Still open</u>: - Can we improve the practical performance of this method?
- Can we improve IPMs further?