Algorithms for Linear Programming

Optimization and Machine Learning Seminar, NTUA December 2018

These notes are intended as reference notes for participants of the Optimization and Machine Learning Seminar in NTUA. Thus, they are in no way complete or free of errors. Please exercise critical thinking and don't take anything for granted.

1. Introduction

Linear Programming (LP) is one of the most widely used primitives that is prevalent both in theory and in practice. Given linear constraints as well as a linear objective, the goal is to find a solution that optimizes the objective while meeting the constraints. The constraints can be inequalities or equalities, but it can be shown that any LP can be brought in the following form:

$$\min_x d^T x$$
$$Bx \geq c$$

where $x, d \in \mathbb{R}^n$, $c \in \mathbb{R}^m$, and $B \in \mathbb{R}^{m \times n}$.

In this lecture we will be looking for "exact" algorithms for solving LPs, in the sense that the error dependence will be $O(\log \frac{1}{\epsilon})$ as opposed to $O(\text{poly}(\frac{1}{\epsilon}))$. For this reason, we are departing from the gradient descent analyses in the previous lectures and look for algorithms with faster convergence. Another reason we are departing from that framework is that we would need to project on the polytope (the linear constraint set), which is a hard problem in general.

2. Cutting plane methods

It is not hard to see that LP solving can be reduced to checking the feasibility of a set of linear inequalities (i.e. halfspaces). A natural strategy to check the feasibility is the following:

- Start with some polytope P that contains the feasible region F of the LP, if it exists (P can be, for example, a ball with very large radius).
- We pick a point p in P and check if p is feasible for the LP. If yes, we are done.
- If not, we find a halfspace H that contains F but not p and update P to $P \cap H$.
- Continue with step 2.

To ensure that our algorithm is making progress, we have to make sure that a large fraction of the polytope is removed in every iteration. Therefore we have to pick point p in such a way that no matter which halfspace is removed, the polytope shrinks by a lot. The following theorem due to Grunbaum exactly answers this problem:

Theorem 2.1. Let K be a convex body, h a hyperplane, and K_1, K_2 the two disjoint parts of the body into which h splits it. If h contains the centroid $\frac{1}{Vol(K)} \int\limits_{x \in K} x dx$ of K, then $Vol(K_1) \leq (1 - \frac{1}{e})Vol(K)$ and $Vol(K_2) \leq (1 - \frac{1}{e})Vol(K)$.

In particular, this implies that if we pick the centroid as the point p in our algorithm, then the volume of the polytope P drops by a constant factor in every iteration. Therefore,

$$Vol(F) \leq Vol(P^k) \leq (1 - \frac{1}{e})Vol(P^{k-1}) \leq \cdots \leq (1 - \frac{1}{e})^k Vol(P^0)$$

and if we have that $Vol(P^0) \leq R^n$ and $Vol(F) \geq r^n$, where n is the dimension of the space, we get convergence after $O(n \log \frac{R}{r})$ iterations.

However, there is an issue here: We don't know how to efficiently maintain the centroid! This means that this algorithm isn't efficient. There are, however, similar algorithms that actually manage instead of keeping the centroid keep some approximate centroid that is easier to compute. For more information see [LSW15].

2.1. **Ellipsoid method.**

The Ellipsoid method [Kha79] essentially instead of maintaining a whole polytope P, maintains an ellipsoid E that shrinks after every iteration. Specifically, we modify the algorithm as follows:

- Start with some ellipsoid E that contains the feasible region F of the LP, if it exists (E can be, for example, a ball with very large radius).
- We pick a point p in E and check if p is feasible for the LP. If yes, we are done.
- If not, we find a halfspace H that contains F but not p and update E to the smallest-volume ellipsoid that contains $E \cap H$.
- Continue with step 2.

As a matter of fact, we can guarantee that the volume of the ellipsoid drops by a factor of $1 - \frac{1}{2n}$ in every iteration, so it drops by a constant factor every $O(n)$ iterations. Therefore the number of iterations is $O(n^2 \log \frac{R}{r})$. Furthermore, each iteration can be computed efficiently (in $O(n^2)$ time) as it is essentially a rank-1 update to the matrix defining the ellipsoid. The point p will still be the centroid, but this is just the center of the ellipsoid and so is trivial to maintain.

2.2. **Solving exponential-sized LPs with cutting plane methods.**

A really interesting property of cutting plane methods is that they don't require an explicit description of all the constraints. The only thing they need is a *separating oracle*, i.e. given an infeasible point p to be able to find efficiently a constraint that is violated by p. As a matter of fact, it turns out that even though the actual polytope might have an exponential number of constraints, cutting planes methods might be able to solve it in polynomial time! One interesting example is the matching polytope: Even though it has been proven that all LP formulations of this problem have exponential size, it can still be optimized using cutting plane methods.

3. Interior Point Methods

Interior Point Methods (IPMs) pick an approach that is fundamentally different from that of Cutting planes methods. Instead of looking for any feasible point, we are using a different formulation in which it is easy to find a feasible point, but the hard part is finding one that optimizes the objective. In fact, IPMs always maintain a feasible point and keep improving its objective value, until it is close to the optimal value.

More specifically, consider the formulation

$$\begin{aligned} \min_x \; & d^T x \\ & Bx \geq c \end{aligned}$$

As we discussed before, an important issue for optimization algorithms here is that projecting onto the constraint set might be hard. The way IPMs deal with this issue is to turn this into an unconstrained minimization problem by incoroprating the constraints into the objective. More specifically, let $\Phi(x)$ be a *barrier* function, i.e. a function defined in the interior of the polytope P, such that $\lim_{x \to \partial P} \Phi(x) = \infty$. In words, this function blows up near the boundary of the polytope. We will modify the objective to contain this term, along with some tradeoff parameter $\mu > 0$:

$$f_\mu(x) = \min_x \, d^T x + \mu\Phi(x)$$

and we will prove that for small enough μ (depending on the desired bound on the error ϵ), the minimizer of this objective will actually be an approximate minimizer of the LP. Our specific choice of barrier function will be the log barrier, i.e.

$$\Phi(x) = -\sum_{i=1}^{m} \log(Bx - c)_i$$

which we will also sometimes write as

$$\Phi(x) = -\sum_{i=1}^{m} \log s_i$$

where $s_i = (Bx - c)_i$ is the *slack* of the i-th constraint in the current solution x. It is easy to see that as some constraint i becomes tighter, $s_i \to 0$, and so $\Phi(x) \to \infty$. Therefore the barrier essentially allows us to make sure that we aren't too close to the boundary so that we don't violate any constraints.

Let's compute the gradient and Hessian of f_μ, where we denote $S = diag(s)$:

$$\nabla f_\mu(x) = d - \mu B^T S^{-1}\vec{1}$$

$$\nabla^2 f_\mu(x) = \mu B^T S^{-2} B$$

As it turns out, even though we don't know of any way of *directly* minimizing $f_\mu(x)$ for a given μ, given the optimal solution for some μ, we can find the optimal solution for some smaller μ' fast. This motivates the following section.

3.1. **Central path.**

Let x^* be the optimal solution of the LP (and for convenience let it be unique) and

$$x^\mu = \operatorname*{argmin}_x \, f_\mu(x)$$

$$s^\mu = Bx^\mu - c$$

As we already mentioned, $\lim_{\mu\to 0} x^\mu = x^*$. The curve traced by x^μ for $\mu > 0$ is called the *central path* and it is a path connecting some kind of "center" of the polytope with the actual optimum. Our goal will be to follow this central path closely until we reach close enough to x^*. The following lemma bounds how small μ has to be.

Lemma 3.1. $d^T x_\mu \leq d^T x^* + \mu m$

Proof. Since x^μ minimizes f_μ, we have $0 = \nabla f_\mu(x^\mu) = d - \mu B^T (S^\mu)^{-1}\vec{1}$. Therefore

$$d^T x^\mu - d^T x^* = \mu\vec{1}^T (S^\mu)^{-1} B(x^\mu - x^*) = \mu\vec{1}^T (S^\mu)^{-1}(s^\mu - s^*) = \mu \sum_i \frac{s_i^\mu - s_i^*}{s_i^\mu} \leq \mu m$$

□

The above lemma implies that computing $x^{\epsilon/m}$ is enough to guarantee ϵ accuracy.

3.2. **Newton's method.** One question that remains is, given x^{μ}, how to quickly compute $x^{\mu'}$ for some $\mu' < \mu$. To this end, we will use Newton's method, which we introduce in this section. Note that Newton's method is a general algorithm and its use is broader than in the context of IPMs.

Remember that we basically derived the gradient descent step by looking at the Taylor expansion of our function:

$$f(x+\delta) = f(x) + \nabla f(x)^T \delta + \frac{1}{2}\delta^T \nabla^2 f(x) \delta + O(\|\delta\|_2^3)$$

where we basically replaced the quadratic term by $\|\delta\|_2^2$ and ignored the higher-order terms. However, the following question arises: Why not keep the quadratic term and take it into account when optimizing, to get a "smarter" step? In fact, we get the following minimization problem:

$$\min_{\delta} \nabla f(x)^T \delta + \frac{1}{2}\delta^T \nabla^2 f(x) \delta$$

which, by taking a first-order optimality condition, gives

$$\nabla f(x) + \nabla^2 f(x)\delta = 0 \Leftrightarrow \delta = -[\nabla^2 f(x)]^{-1} \nabla f(x)$$

this is exactly one step of Newton's method, i.e.

$$x^{t+1} = x^t - [\nabla^2 f(x)]^{-1} \nabla f(x)$$

In general, the guarantees of Newton's method are of similar nature to the gradient descent guarantees we discussed in the last lectures. However, for an interesting class of functions called *self-concordant*, Newton's method has the interesting property that if it is already relatively close to the optimum, convergence is *extremely* fast.

Before attempting to analyze it we need to set up some appropriate definitions in order to provide us with the quantities that make sense to keep track of:

Definition 3.2. Given a function $f : \mathbb{R}^n \to \mathbb{R}^n$, the *local norm* of $u \in \mathbb{R}^n$ at x is defined as

$$\|u\|_x = \|u\|_{\nabla^2 f(x)} = \sqrt{u^T \left[\nabla^2 f(x)\right] u}$$

and the *dual local norm* of $v \in \mathbb{R}^n$ at x is defined as

$$\|v\|_x^* = \|v\|_{[\nabla^2 f(x)]^{-1}} = \sqrt{v^T \left[\nabla^2 f(x)\right]^{-1} v}$$

Definition 3.3. $f : \mathbb{R}^n \to \mathbb{R}^n$ is called *self-concordant* if for every 1-dimensional restriction $g : \mathbb{R} \to \mathbb{R}$ of f,

$$|g'''(x)| \leq 2g''(x)^{3/2}$$

This implies the following useful property for the Hessian of f

$$\nabla f(x) \left(\frac{1}{1 + \|\nabla f(x)\|_x^*}\right)^2 \preceq \nabla^2 f(x+\delta) \preceq \nabla f(x) \left(\frac{1}{1 - \|\nabla f(x)\|_x^*}\right)^2$$

for any $\delta \in \mathbb{R}^n$.

The potential we will keep track of for Newton's method will be $\|\nabla f(x)\|_x^*$, i.e. the dual local norm of the gradient. In fact, we will show the following:

Lemma 3.4. If f is self-concordant and $\|\nabla f(x^0)\|_{x^0}^* \leq \frac{1}{10}$, Newton's method converges to an ϵ-approximate optimum in $O(\log\log\frac{1}{\epsilon})$ iterations.

Proof. Consider a Newton step $x' = x - [\nabla^2 f(x)]^{-1}\nabla f(x)$. Let's denote $H = \nabla^2 f(x)$ and $H' = \nabla^2 f(x')$. From the self-concordance property, we have that $\frac{1}{2}H \preceq H' \preceq 2H$. Now, the mean value theorem implies that $\nabla f(x') - \nabla f(x) = \bar{H}(x'-x)$, and together with the self-concordance property

$$H\left(\frac{1}{1+\|\nabla f(x)\|_x^*}\right)^2 \preceq \bar{H} \preceq H\left(\frac{1}{1-\|\nabla f(x)\|_x^*}\right)^2$$

which, by simplifying because $\|\nabla f(x)\|_x^* \leq \frac{1}{10}$ implies

$$H\left(1-3\|\nabla f(x)\|_x^*\right) \preceq \bar{H} \preceq H\left(1+3\|\nabla f(x)\|_x^*\right)$$

So now

$$\begin{aligned}
\|\nabla f(x')\|_{x'}^* \leq & 2\|\nabla f(x')\|_x^* \\
= & 2\|\nabla f(x) + \bar{H}(x'-x)\|_{H^{-1}} \\
= & 2\|(I - \bar{H}H^{-1})\nabla f(x)\|_{H^{-1}} \\
= & 2\|(I - H^{-1/2}\bar{H}H^{-1/2})H^{-1/2}\nabla f(x)\|_2 \\
\leq & 2\|I - H^{-1/2}\bar{H}H^{-1/2}\|_{2\to 2}\|H^{-1/2}\nabla f(x)\|_2 \\
\leq & 2\|I - H^{-1/2}\bar{H}H^{-1/2}\|_{2\to 2}\|\nabla f(x)\|_x^*
\end{aligned}$$

where $\|A\|_{2\to 2} = \max\limits_{u\in\mathbb{R}^n}\frac{\|Au\|_2}{\|u\|_2}$, or just the maximum eigenvalue of A. From the relation between $\bar{H}$ and H that comes from self-concordance we get

$$-3\|\nabla f(x)\|_x^* \preceq I - H^{-1/2}\bar{H}H^{-1/2} \preceq 3\|\nabla f(x)\|_x^*$$

and so

$$\|I - H^{-1/2}\bar{H}H^{-1/2}\|_{2\to 2} \leq 3\|\nabla f(x)\|_x^*$$

Putting everything together,

$$\|\nabla f(x')\|_{x'}^* \leq 6\left(\|\nabla f(x)\|_x^*\right)^2$$

If we denote $\lambda_t = \|\nabla f(x^t)\|_{x^t}^*$, we have

$$\lambda_k \leq 6\lambda_{k-1}^2 \leq \cdots \leq 6^{2^k-1}\lambda_0^{2^k}$$

and so for $k = \Theta(\log\log\frac{1}{\epsilon})$ we get $\lambda_k \leq \epsilon$. □

3.3. **IPM analysis.** First of all, our function $f_\mu(x) = d^Tx - \sum\limits_i \log s_i$ is self-concordant. As we mentioned before, we will use Newton's method to find $x^{\mu'}$ given x^μ. Let $\mu' = \mu(1-\eta)$. To find out how large η can be, we compute

$$\begin{aligned}
\|\nabla f_{\mu'}(x^\mu)\|_{x^\mu}^* &= \|c - \mu' B^T(S^\mu)^{-1}\vec{1}\|_{x^\mu}^* \\
&= \|c - \mu B^T(S^\mu)^{-1}\vec{1} + \delta B^T(S^\mu)^{-1}\vec{1}\|_{x^\mu}^* \\
&= \|\delta B^T(S^\mu)^{-1}\vec{1}\|_{x^\mu}^* \\
&= \delta\sqrt{\vec{1}^T(S^\mu)^{-1}B(B^T(S^\mu)^{-2}B)^{-1}B^T(S^\mu)^{-1}\vec{1}} \\
&\leq \delta\sqrt{\vec{1}^T\vec{1}} \\
&= \delta\sqrt{m} \\
&\leq \frac{1}{10}
\end{aligned}$$

for $\delta \leq \frac{1}{10m}$. The second to last inequality comes from the fact that

$$(S^{\mu})^{-1}B(B^T(S^{\mu})^{-2}B)^{-1}B^T(S^{\mu})^{-1}$$

is an orthogonal projection matrix and thus contracts the ℓ_2 norm.

Therefore we can take $\mu' = \mu(1 - \frac{1}{10\sqrt{m}})$ and since we will be decreasing μ by a factor of poly$\left(\frac{m}{\epsilon}\right)$ over the whole course of the algorithm, the total number of iterations will be $O(\sqrt{m}\log\frac{m}{\epsilon})$. This algorithm was devised by Renegar in 1988 [Ren88].

References

[Kha79] Leonid G Khachiyan. A polynomial algorithm in linear programming. In *Doklady Academii Nauk SSSR*, volume 244, pages 1093–1096, 1979.

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 1049–1065. IEEE, 2015.

[Ren88] James Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical Programming*, 40(1-3):59–93, 1988.