Algorithms for Convex Optimization

1 Simplex Algorithm

1.1 Background

The Simplex Algorithm, due to Dantzig in 1947, was the first methodical procedure for solving linear programs. It includes a family of algorithms parametrized by a **pivot rule** (essentially a choice of tie-breaker rules). The algorithm is efficient in practice, but has exponential time complexity in theory; however, this method spurred the development of the theoretically efficient ellipsoid algorithm.

Consider an LP with n variables and m constraints, with the form

$$\begin{aligned} \max \ & \boldsymbol{c}^\intercal \boldsymbol{x} \\ \text{s.t. } & \mathbf{A}\boldsymbol{x} \preceq \boldsymbol{b} \end{aligned}$$

Recall that the optimal solution to this LP occurs at a vertex of the feasible polytope, and corresponds to n linearly independent tight inequalities. Assuming nondegeneracy (there is no vertex with greater than n such inequalities), we can apply the Simplex Algorithm.

1.2 Algorithm

The Simplex Algorithm is divided into two phases. In the first phase, we select a starting vertex $\boldsymbol{x}_0$. In the second phase, we use $\boldsymbol{x}_0$ to compute the optimal vertex $\boldsymbol{x}^*$. At each step of the second phase algorithm, we move to a neighboring vertex $\boldsymbol{x}' = \boldsymbol{v} + \boldsymbol{x}$ with $\boldsymbol{cx}' > \boldsymbol{cx}$. Then, either $\boldsymbol{c}$ is in the cone defined by the tight constraints at $\boldsymbol{x}'$, implying $\boldsymbol{x}^* = \boldsymbol{x}'$ by complementary slackness, or we can improve $\boldsymbol{cx}$ by moving along an edge of the polytope. Incidentally, this algorithm will also produced the optimal dual $\boldsymbol{y}^*$; in this view, the Simplex Algorithm is a primal-dual algorithm.

1.3 Properties

Claim 1. If the Simplex Algorithm terminates, then it will correctly output a primal-dual pair or correctly output that the LP is unbounded.

Note that the primal feasibility of $\boldsymbol{x}$ is maintained throughout, and $(\boldsymbol{x}, \boldsymbol{y})$ is returned only if $\boldsymbol{y}$ is the dual feasible which satisfies complementary slackness. If so, then by strong duality $\boldsymbol{x}$ and $\boldsymbol{y}$ are both optimal. Furthermore, the Simplex Algorithm returns that the LP is unbounded only if $\exists \boldsymbol{d}$ with $\boldsymbol{c}^\intercal \boldsymbol{d} > 0, \mathbf{A}\boldsymbol{d} \preceq 0$, which is the definition of unboundedness.

Claim 2. Nondegeneracy implies that the Simplex Algorithm terminates in at most $\binom{m}{n} \leq 2^m$ steps.

This is because the LP polyhedron has at most $\binom{m}{n}$ vertices and (by nondegeneracy) the algorithm never repeats a vertex. Also, the objective strictly improves at each vertex, and it is bounded, so the algorithm will find the best vertex after checking at most $\binom{m}{n}$ of them.

Algorithm 1 Simplex Algorithm

```
Write $\boldsymbol{x} = \boldsymbol{x}_0$.
while true do
  Let $T$ be the set of linearly independent rows of $\mathbf{A}$ which are tight at $\boldsymbol{x}$. Write $\boldsymbol{c} = \mathbf{A}_T^\intercal \boldsymbol{y}_T$ for such a $T$.
  These are exactly the dual constraints if we include $\boldsymbol{y} \succeq 0$.
  if $\boldsymbol{y} \succeq 0$ (i.e., the dual constraints are satisfied) then
    return $(\boldsymbol{x}, \boldsymbol{y})$
  else
    Choose $i$ with $y_i < 0$, and let $\boldsymbol{d}$ be such that $\mathbf{A}_{T\setminus\{i\}}^\intercal \boldsymbol{d} = 0$ and $a_i \boldsymbol{d} = -1$. That is, we choose
    a constraint $i$ such that moving in the direction $\boldsymbol{d}$ preserves tightness of constraints in $\mathbf{A}_{T\setminus\{i\}}$ and
    loosens constraint $i$. Because $\mathbf{A}_T$ is full rank, we can choose any $\boldsymbol{d} \in null(\mathbf{A}_{T\setminus\{i\}})$. This improves
    the objective because $\boldsymbol{c}^\intercal \boldsymbol{d} = \boldsymbol{y}^\intercal \mathbf{A}\boldsymbol{d} = y_i a_i \boldsymbol{d} > 0$.
    if $\boldsymbol{x} + \lambda \boldsymbol{d}$ feasible for all $\lambda > 0$ then
      return unbounded
    end if
    Set $\boldsymbol{x} = \boldsymbol{x} + \lambda \boldsymbol{d}$ for the largest $\lambda$ maintaining feasibility. By nondegeneracy, $\lambda > 0$, so we essentially
    select a new tight constraint $j$ to replaced the loosened constraint $i$.
  end if
end while
```

1.4 Pivot Rules

The Simplex Algorithm seems to work well so far. However, nondegeneracy is a strong assumption. We need a rule to answer which vertex we choose when multiple vertices improve the objective, and how to identify the next vertex in the presence of degeneracy. In particular, degeneracy allows for several algebraic vertices to represent the same geometric vertex, so we need to avoid choosing an equivalent algebraic vertex to guarantee termination. Thus we can define a pivot rule, which specifies the order in which we examine algebraic representations of vertices (that is, which constraint i will leave T and which constraint j will enter T). Some examples include:

1. Bland's Rule: Select the lowest indexed i and the lowest indexed j.
2. Lexicographic: Maintain an order over the rows of T, then move from T to the lowest ordered T'.
3. Perturbation: Perturb b by small values to remove degeneracy (this can be a symbolic perturbation).

Many pivot rules have been shown to never repeat an algebraic vertex, but no pivot rules have been shown to guarantee a polynomial number of checked points. Thus, the Simplex Algorithm has exponential worst-case runtime. However, that doesn't explain why the algorithm is so efficient in practice. In 2001, Shang-Hua Teng and Daniel Spielman invented the field of smoothed analysis, which analyzes algorithms with noise representing measurement error. It turns out that the Simplex Algorithm has polynomial smoothed complexity which scales with the inverse of the standard deviation of the measurement error – so in many real-life cases where measurement error is small, the Simplex Algorithm runs efficiently.

Open Question: Is there a pivot rule which guarantees a polynomial number of pivots in the worst case? A solution to this question would yield a strongly polynomial time algorithm for solving LPs (i.e., one that does not depend on the desired bit precision). It would also resolve a classic open question in polyhedral combinatorics: the Polynomial Hirsch Conjecture.

1.5 Initialization

We still need to figure out how to select our starting vertex $\boldsymbol{x}_0$. For this, we design an LP whose optimal solution is a vertex of the polyhedron formed by the constraints of the original LP. First note that if $\boldsymbol{x} = \mathbf{0}$

is feasible in the original LP, then it must be a vertex, and we can use that to start. Otherwise, we design a new LP with a variable z measuring how far a solution is from feasibility:

$$\begin{aligned} &\min z \\ &\text{s.t } \mathbf{A}\boldsymbol{x} - z\mathbf{1} \preceq \boldsymbol{b} \\ &\quad \boldsymbol{x} \succeq 0 \\ &\quad z \geq 0 \end{aligned}$$

If the original LP is feasible, then $z = 0$ is OPT for the new LP. So, the optimal vertex solution with $z = 0$ corresponds to the vertex $\boldsymbol{x}_0$ of the original LP. This LP is solvable in polynomial time by applying Phase II of the Simplex Algorithm using $\boldsymbol{x}_0 = \mathbf{0}, z_0 = -\boldsymbol{b}_{min}$.

2 Ellipsoid Algorithm

The Ellipsoid Algorithm was developed in 1976 by Yudin, Nemirovski, and Shor as a heuristic for convex optimization; then in 1979, Khachiyan showed that it yields a polynomial time algorithm for solving LPs, a significant theoretical breakthrough. Though it is inefficient in practice, it is theoretically powerful and has deep consequences for complexity and optimization. The Ellipsoid Algorithm solves the **convex feasibility problem**: Given a description of a convex set $K \subseteq \mathbb{R}^n$ (either directly or by a separation oracle), an ellipsoid $E(\boldsymbol{c}, \mathbf{Q})$ containing K, a rational number $R > 0$ with $vol(E) \leq R$, and a rational number $r > 0$ with $vol(K) \geq r$, find a point $\boldsymbol{x} \in K$ or else declare that K is empty. The intuition here is that the Ellipsoid Algorithm performs binary search in high dimension.

Algorithm 2 Ellipsoid Algorithm

```
$E = E(\boldsymbol{c}, \mathbf{Q}) \supseteq K$
while true do
    Using the separation oracle, check if $\boldsymbol{c} \in K$
    if $\boldsymbol{c} \in K$ then
        Output $\boldsymbol{c}$
    else
        Oracle returns a separating hyperplane $\boldsymbol{h}$ such that $K$ is contained in the half-ellipsoid $E \bigcap \{\boldsymbol{y} : \boldsymbol{h}^\intercal \boldsymbol{y} \leq \boldsymbol{h}^\intercal \boldsymbol{c}\}$
    end if
    Let $E' = E(\boldsymbol{c}', \mathbf{Q}')$ be the minimum volume ellipsoid containing the half-ellipsoid above
    if $vol(E') \geq r$ then
        Set $E = E'$
    else
        Return empty
    end if
end while
```

Claim 3. If the Ellipsoid Algorithm terminates, then it either outputs an $\boldsymbol{x} \in K$ or correctly outputs that K is empty.

We know that the algorithm only outputs $\boldsymbol{x}$ if the oracle confirms that it is in K. $E \supseteq K$ is maintained throughout the algorithm by definition, and we are promised that $vol(K) < r$ iff $K = \emptyset$. Thus, if the algorithm outputs empty then $r > vol(E) \geq vol(K)$ and thus $K = \emptyset$.

Theorem 4. *The Ellipsoid Algorithm runs in polynomial time.*

Proof.

Lemma 5. *The minimum volume ellipsoid containing half of another ellipsoid can be computed in polynomial time. Also, the volume of the new ellipsoid is smaller than the volume of the previous ellipsoid by a factor of at least* $\exp(\frac{1}{2(n+1)}) \approx 1 + \frac{1}{2(n+1)}$*. There is a tricky-to-prove, easy-to-compute closed form for the matrix and center of the new ellipsoid in terms of the matrix and center of the previous ellipsoid.*

The volume of the working ellipsoid starts off at R. It decreases by a factor of $\exp(\frac{1}{2(n+1)})$ each iteration. Therefore after $2(n+1)\ln R/r$ iterations, the volume is less than r, triggering termination. So, in each iteration we 1) call the separation oracle, 2) compute the minimum-volume ellipsoid, and 3) compute the volume of the new ellipsoid. Step 1 is polynomial time by definition, step 2 is polynomial time by the lemma, and step 3 is polynomial time because it is a determinant computation. Thus the Ellipsoid Algorithm runs in polynomial time. $\square$