**CMPSCI 611 Advanced Algorithms** **Fall 2000**

# Lecture 25: December 12

*Lecturer: Micah Adler* *Scribe: Marc Pickett I*

This lecture investigates the Simplex algorithm, a popular method for solving large linear programming problems.

## 25.1 Linear Programming

Given a linear programming problem (with $M$ equations (or inequalities) of $N$ variables), we can construct a vector $C$ of length $n$, an $m$ by $n$ matrix $A$, and a vector $b$ of length $m$, such that our goal is to find a vector $x$ of length $n$ such that $Ax = b$, $x_i \geq 0$, and $Cx$ is as small as possible. ($m$ and $n$ will be $O(M)$ and $O(N)$, respectively.)

Assuming a solution exists, let $F$ be the feasible set of solutions to $Ax = b$. Then $F$ can be represented as a polyhedron in $\Re^{n-m}$, and either $Cx$ will be minimized at a vertex of $F$, or the solution will be unbounded.

One naive approach to finding the solution is to simply search through all the vertices of $F$, but there are an exponential number of vertices (in $n$). Are there better methods for finding the solution?

## 25.2 The Simplex Algorithm

One method is the Simplex algorithm. The intuitive idea is to pick a starting vertex $x_1$, look at its neighbors, and then move to a neighbor $x_2$ such that $Cx_2 < Cx_1$. If no such neighbor exists, you've found your solution.

## 25.3 Finding Neighbors

A vertex is an intersection of at least $n-m$ facets of the polyhedron (since $Ax = b$ has solutions where $n-m$ variables of $x$ are set to 0). For now, let's assume that the vertices are at the intersections of exactly $n-m$ facets. So there will be $m$ variables of $x$ that are nonzero.

Let's specify the indices of these vertices by a vector $B$, such that $x_{B_1}, x_{B_2}, \cdots, x_{B_m}$ are all nonzero. Now, in the expression $Ax$, only the columns $A_{B_1}, A_{B_2}, \cdots, A_{B_m}$ will be relevant because entries in the other columns will always be multiplied by 0.

Let $\hat{A}$ be the matrix made by removing the irrelevant columns from $A$, i.e., let $\hat{A} = [A_{B_1}, A_{B_2}, \cdots, A_{B_m}]$ (where $A_i$ is the $i$th *column* of $A$). Likewise, let $\hat{x}$ be the vector made by removing the irrelevant entries from $x$ (i.e., $\hat{x} = [x_{B_1}, x_{B_2}, \cdots, x_{B_m}]^T$).

Now, we can simplify our equation: $\hat{A}\hat{x} = b$. Using this, we can quickly solve for $x$: $\hat{B}^{-1}b = \hat{x}$. Note that this solution may have values of $\hat{x} < 0$ if we pick the wrong values for $\hat{B}$.

To move from 1 vertex to another, we just need to change the values of $B$ 1 at a time (i.e., exchange them). That is, let $B'_i = B_i$ for all $i \neq j$ for some $k$, and set $B'_k$ to be a new index (that isn't already taken by any $B'_i$ or equal to $B_k$). We can find $x$'s neighbor $x'$ by solving for $\hat{A}'^{-1}b' = \hat{x}'$.

## 25.4 Finding the Neighbor's Value Quickly

Because we've already solved for $\hat{x}$, we know already that $\sum_{i=1}^{m} x_{B_i} A_{B_i} = b$. Suppose we let $x_j$ be the new nonzero variable (in $x'$).

Because of assumptions of the columns of $A$, $A_j$ can be represented as a linear combination of $A_{B_i}$. Let $t_{ij}$ be the terms in this combination (so that $A_j = \sum_{i=1}^{m} t_{ij} A_{B_i}$). If we set $x_j = \lambda$ then

$$\begin{aligned}
b &= \sum_{i=1}^{m} x_{B_i} A_{B_i} && (25.1)\\
&= \lambda A_j + \left(\sum_{i=1}^{m} x_{B_i} A_{B_i}\right) - \lambda A_j && (25.2)\\
&= \lambda A_j + \left(\sum_{i=1}^{m} x_{B_i} A_{B_i}\right) - \lambda \left(\sum_{i=1}^{m} t_{ij} A_{B_i}\right) && (25.3)\\
&= \lambda A_j + \sum_{i=1}^{m} \left(x_{B_i} - \lambda t_{ij}\right) A_{B_i} && (25.4)
\end{aligned}$$

Therefore, to make another term 0, we just increase $\lambda$ until $x_{B_i} = \lambda t_{ij}$ for some $i$. That is, we can solve $\lambda_0 = \min_i \frac{x_{B_i}}{t_{ij}}$. If $t_{ij}$ is non-positive, then we just ignore that index $i$ (and move on to the next index). If $t_{ij} \leq 0$ for all $i$ and $j$, then the vertex is unbounded.

To find the best neighbor to expand, pick $l$ such that $B_l = j$. Let $x' = x_{b_i} - \lambda_0 t_{ij}$ if $i \neq l$, and let $x' = \lambda_0$ if $i = l$. Now pick $l$ so that the neighbor $x'_l$ has the minimum value of $Cy_l$.

## 25.5 Runtime

Using a data structure called a "tableau", one can move from 1 vertex to the next in $O(nm)$. Unfortunately, no one has found a way to pick pivots that guarantees convergence in polynomial time. However, in most practical cases, the simplex algorithm takes about linear time. A few different techniques are shown to be polynomial: Khachiyan's [K79] is usually impractical, Karmarkar's [K84] is more practical.

For a nice description of Karmarkar's algorithm (as well as the Simplex algorithm), please see [S88].

## References

K84 N. Karmarkar, A New Polynomial-Time Algorithm for Linear Programming, *Combinatorica* 4, 1984, pp. 373-395.

K79 L.G. Khachiyan, A Polynomial Algorithm for Linear programming, *Soviet Mathematics Doklady* 20, 1979, pp. 191-194.

S88 G. Strang, *Linear algebra and its applications*, Harcourt Brace and Co, 1988.