Lectures 3 and 4

Uri Feige

May 1 and May 8, 2002

1 The simplex algorithm

The simplex algorithm was designed by Danzig in 1947. We present the main ideas involved.

1.1 A geometric view

Recall that a linear program defines a polyhedron. For simplicity, let us assume here that this polyhedron is nonempty (i.e., the LP is feasible) and bounded (namely, it is a polytope). Then we know that the optimal value of the LP is attained at a vertex of the polytope (equivalently, at a basic feasible solution to the LP). We say that two vertices of a polytope are adjacent if they are connected by an edge of the polytope.

The basic idea of the simplex algorithm is as follows. One starts at an arbitrary vertex of the polytope. (The question of how to find a starting vertex will be addressed shortly.) Thereafter, at every iteration, the algorithm moves to a neighboring vertex of better value (of the objective function). The algorithm ends when such a move is no longer possible, meaning that the current vertex is a local optimum compared to all adjacent vertices.

The above description gives only the basic idea. A more serious treatment invloves proving that the solution found is optimal, showing how one can implement a single iteration, deciding which of several improving adjacent vertices to move to, how to find a starting feasible vertex, deal with polyhedrons that are unbounded, with degeneracies, analysing the number of iterations required (and showing that this number is finite) and so on. An even more serious treatment involves the many ideas that come into improved implementations of the simplex algorithm.

1.2 LPs in various forms

In principle, a simplex-like algorithm can be run on linear programs in cannonical form, or even in general form. However, for reasons of efficiency, the simplex algorithm is run on linear programs in standard form. This allows for many shortcuts in the implementation.

We note however that there are cases where one runs a simplex-like algorithm on an LP that is not in standard form. This may happen if the number of constraints is much larger than the number of variables. Transforming such an LP to standard form greatly increases the number of variables (by introducing a slack variable for every constraint), an effect that is undesirable. Moreover, in some cases the number of constraints is so large that they are not explicitly recorded. Instead, they are generated "on the fly" during the run of the algorithm, making it impossible to work in standard form. (Later in the course, when we discuss the Ellipsoid algorithm, we shall see scenarios where constraints are not given explicitly.)

1.3 Finding a feasible solution

In order to start the simplex algorithm, one needs some basic feasible solution. In principle, the complexity of finding whether an LP is feasible is polynomially related to that of finding the optimal solution to LPs. Given the ability to find optimal solutions, we can certainly find feasible solutions. Given the ability to find feasible solutions, we can find optimal solutions by performing binary search on the value of the objective function (treated as a constraint). Here we use the fact that the optimal solution, if bounded, is attained at a vertex, and that the numerical precision (representing numbers as rationals) that is polynomially related to the input size suffices in order to exactly represent a vertex.

Hence in principle, one should not expect to be able to find feasible solutions significantly more quickly than optimal solutions, and the problem of starting the simplex algorithm is as hard as the problem of running it. This gives a clue as to how to start the simplex algorithm.

Recall that we are dealing with LPs in standard form.
minimize $c^T x$
suject to
$Ax = b$
$x \geq 0$

We may assume without loss of generality that $b \geq 0$, as we can always multiply constraints by -1. Now introduce a vector y of m variables (similar to slack variables) and consider the new LP in standard form:
minimize $1^T y$
suject to
$Ax + y = b$
$x \geq 0$
$y \geq 0$

(Here 1^T is the all 1 row vector.) In this new program, setting $x = 0$ and $y = b$ is a bfs. Hence the simplex algorithm can be started. Let x^*, y^* be the final bfs found by the simplex algorithm, and assume that it is optimal. There are three cases:

1. $y^* = 0$ and x^* has exactly m nonzero coordinates. In this case x^* is a bfs for the original program, and the nonzero variables are the basic variables.

2. $y^* = 0$ and x^* has less than m nonzero coordinates. In this case x^* is a bfs for the original program, and one can complete the set of nonzero variables to a basis (by adding 0-variables whose respective columns are linearly independent).

3. $y^* \neq 0$. In this case the original LP is not feasible.

Hence to start the simplex algorithm on an LP, one first runs the simplex algorithm on an initialization LP.

1.4 The linear algebra of the simplex algorithm

Consider a linear program in standard form.

minimize $c^T x$

suject to

$Ax = b$

$x \geq 0$

We assume at this point that there are no degeneracies (every bfs m has nonzero variables), and that we are given some bfs. In this case, a single iteration of the simplex algorithm will do the following:

Move to an adjacent bfs with lower value of $c^T x$. If no such adjacent bfs exists, stop.

We shall show now that such an iteration can be performed in polynomial time.

Let $x_B = x_{B(1)}, \ldots, x_{B(m)}$ be the current bfs, and let $B = (A_{B(1)} \ldots A_{B(m)})$ be the corresponding basis matrix.

Let x_j be a nonbasic variable, and lets check whether it is profitable to have it enter the basis.

We know that at the beginning of the current iteration,

$Bx_B = Bx_B + A_j x_j = b$.

Define direction variables d_B, d_j that indicate by how much we change the current value of the basic variables and of x_j. (For other variables, their corresponding value in the vector d is 0.) Then they need to satisfy $Bd_B + A_j d_j = 0$, implying

$d_B = -B^{-1} A_j d_j$.

When $d_j = 1$, the change in the objective function is

$\bar{c_j} = c_j - c_B^T B^{-1} A_j$,

which is called the *reduced cost.*

We want x_j to enter the basis only if the reduced cost is negative. We remark here that if x_j is already in the basis, $B^{-1} A_j$ is just the indicator vector for x_j, implying that $\bar{c_j} = 0$.

If the reduced cost is negative, we can move in the direction of d to a distance θ, until one of the previous basic variables becomes 0. Hence:

$\theta = \min_{i \in \text{basis}} -\frac{x_i}{d_i}$.

The x_i for which θ is realized leaves the basis.

If no $d_i < 0$, then the optimum is $-\infty$.

The issue of degeneracies may cause problems (force $\theta = 0$), and they will be addressed at a later section.

Lemma 1 *The new solution reached by one iteration of the simplex algorithm is also a bfs.*

Proof: If not (meaning that the corresponding columns have a linear dependency) then in the vector d leading to it we would necessarily have had $d_i = 0$ for the variable that left the basis, which is a contradiction. $\square$

Lemma 2 *Let x be a bfs with basis matrix B (and matrix N corresponds to the rest of A), and let $\bar{c}$ be the corresponding vector of reduced costs for all nonbasic variables. Then*

1. *If x is optimal and nondegenerate, then $\bar{c} \geq 0$.*
2. *If $\bar{c} \geq 0$, then x is optimal.*

Proof: To prove 1, observe that if $\bar{c_j} < 0$, then moving in the direction of the corresponding d reduces the objective function.

To prove 2, let y be an arbitrary feasible solution, and define $d = y - x$. Then $Ad = 0$, implying $Bd_B + Nd_N = 0$, and $d_B = -B^{-1}Nd_N$. Now we can compute the change in cost that results from a move by d.

$c^T d = c_B^T d_B + c_N^T d_N = (c_N^T - c_B^T B^{-1} N) d_N = \bar{c_N}^T d_N$

As $d_N \geq 0$, and we assumed that $\bar{c_N} \geq 0$, the change in cost is positive. $\square$

1.5 Handling degeneracies

If a bfs is degenerate, than we may be required to choose $\theta = 0$ and there is no progress in the objective function in a single iteration. In this case, we do perform a change of basis (but stay in the same vertex of the polyhedron).

A problem that might occur is *cyclying*. We may continue changing bases in a cyclic function without ever leaving the vertex (even if the vertex is not optimal). One can design examples were cycling occurs, and it has been reported to occur in practice. There are several ways for avoiding cycling:

1. A generic way. A degeneracy is a result of numerical coincidence. Slightly perturbing b at random will eliminate it. However, this approach is not favored because it is computationally costly – requires high precision. Moreover, care must be taken not to ruin feasiblity of the LP.

2. At a degenerate vertex, decide at random which of several plausible nonbasic variables enters the basis, and which plausible basic variable leaves the basis. Eventually, with probability 1, one gets out of the vertex.

3. Use some deterministic pivoting rule that ensures getting out of the vertex. This is the preferred approach. One such rule is Bland's rule: when there is a choice between different variables (to enter or leave the basis), always choose the lexicographically first one.

1.6 Pivot selection

Many nonbasic variables may have negative reduced cost. Hence one needs a rule to resolve the ambiguity of the simplex algorithm. Many options are possible here. Among them we have:

1. Choose the variable with most negative reduced cost.

2. Choose the variable with greatest impact on the objective function (minimizing $\theta \bar{c_j}$).

3. Choose an anticycling rule.

The performance of the simplex algorithm and the complexity of implementing it depends on the particular pivot selecting rule. A major open question asks whether there is a pivot selection rule that guarantees a polynomial number of iterations. We shall discuss this question later in the course.

In practice, in most cases and with many of the pivot selection rules, the number of iterations of the simplex algorithm is typically $O(m)$.

1.7 Implementation issues

The way we described the simplex algorithm appears to invert a matrix B in every iteration, giving a complexity of roughly m^3 operations per iteration. Moreover, potentially one needs to compute $B^{-1}A_j$ for every nonbasic variable, making the complexity $O(m^2n)$. However, we note that between consecutive iterations, the matrix B changes by only one column, a fact that can be used in order to speed up the algorithm. We illustrate his by presenting the *tableau* version of the simplex algorithm.

Let us first recall how Gaussian elimination computes the inverse of a matrix B. One appends to it the identity matrix I to obtain a matrix $C = [I; B]$. Then one performs row operations until the submatrix B is transformed into a matrix I, and at this point the result is $[B^{-1}; I]$. A very useful fact to note is that performing these row operations is equivalent to multiplying C on the left by B^{-1}. (Every row operation is equivalent to multiplying by a matrix, and B^{-1} is the unique solution to the matrix equation $X[I; B] = [B^{-1}; I]$.) Hence if one would append a column y to the matrix C, then the same row operations on the matrix $[I; B; y]$ would lead to the matrix $[B^{-1}; I; B^{-1}y]$. If one is not explicitly interested in the matrix B^{-1}, but only in the product $B^{-1}b$, then there is no need to carry around the first component of the matrix, and it suffices to consider a matrix $[B; y]$ and perform on it row operations until the B changes to I.

Now lets get back to the simplex algorithm. Recall that we wish to compute quantities such as $x_B = B^{-1}b$, and that d_B depends on $B^{-1}A_j$ (where A_j is a column corresponding to a nonbasic variable). Recall that we may view A as $[B; N]$. Consider now the matrix $[B; N; b]$. Performing row operations we can obtain the matrix $[I; B^{-1}N; B^{-1}b]$. Altogether, the number of row operations performed to obtain this matrix is $O(m^2)$, giving a total of $O(m^2n)$ basic operations.

Now we reach the main point. In the next iteration of the simplex algorithm, only one column is exchanged between B and N. Exchanging the locations of these columns in the matrix $[I; B^{-1}N; B^{-1}b]$ gives a matrix whose left portion is identical to I, except for one column. Now m row operations suffice in order to reach the next $[I; B^{-1}N; B^{-1}b]$. Hence the number of operations per iteration can be reduced to $O(mn)$.

It is convenient to add a row $[c_B^T; c_N^T; 0]$ to the matrix above. Performing row operations that make the first component of this row 0, one obtains $[0; c_N^T - c_B^T B^{-1}N; -c_B^T B^{-1}b]$, which gives the reduced costs and the value of the objective function (negated).

Summarizing, when working in tableau form we do the following:

1. For simplicity, think of the columns of A as being rearranged so that the basic variables are first. (This need not be done in practice.) Then we have the matrix $[B; N; b]$. Extend it by the cost row $[c_B^T; c_N^T; 0]$.

2. Perform row operations until one gets the matrix $X = [I; B^{-1}N; B^{-1}b]$. Then perform row operations to modify the cost row to $[0; c_N^T - c_B^T B^{-1}N; -c_B^T B^{-1}b]$.

3. Find a column j with negative reduced cost. (If there are several such columns, use your pivot selection rule to decide among them.) If there is no such column, stop, and then the last column of the matrix gives the bfs and the (negation of the) corresponding value of the objective function.

4. If $X_{ij} < 0$ for all $1 \leq i \leq m$ then stop and report that the objective function is unbounded.

5. Compare the last column to column j, to find $\min[X_{i(n+1)}/X_{ij}]$ over all i with positive X_{ij}. (If there are several such i, use your pivot selection rule to decide among them.)

6. Exchange columns i and j and go back to step 2. (The exchange need not be done explicitly in practice.)

A toy example of how the simplex algorithm is run will be given in class, but is not presented here due to my lazyness.

In practice, one rarely runs the tableau version of the simplex algorithm. A major reason for this is that this version fails to capitalize on special structure of the matrix A.

- If A has a very large aspect ratio $n \gg m$, then one uses techniques that do not hold all of N simultaneously, but just enough so as to find a variable to pivot on.
- Often the matrix A is very sparse (most entries are 0). The tableau version destroys this property. Other versions of the simplex algorithm, like the so called *revised* version, take advantage of sparsness and reduce the number of operations per iteration.
- Sometimes the matrix A has special structure that allows one to decompose it into smaller blocks, and then one runs versions of the simplex algorithm that take advantage of the block structure.

1.8 Column geometry

We give here another geometric view of the simplex algorithm.

$k+1$ vectors $y_1, \ldots, y_{k+1}$ in R^n are affinely independent if the k vectors $(y_i - y_{k+1})$ for $1 \leq i \leq k$ are linearly independent. The convex hull of $y_1, \ldots, y_{k+1}$ in R^n is called a k-dimensional simplex.

Consider an arbitrary linear program in standard form:

minimize $c^T x$

suject to

$Ax = b$

$x \geq 0$

We wish it to have some additional properties, that will be explained shortly. We describe a transformation that allows us to achieve these properties.

Let M be large enough so that for every basic feasible solution $\sum x_i < M$. Let us add an auxiliary slackness variable $y \geq 0$, and the constraint $y + \sum x_i = M$. Dividing all constraints by M, and introducing a new variable z for the objective function, we may bring the LP to the following form:

minimize z

suject to

$Ax = b$

$c^T x = z$

$\sum x_i = 1$

$x \geq 0$

The last two constraints can be viewed as convexity constraints. They require the vector b to be a convex combination of the columns of A (rather than just a nonnegative linear combination), and the objective function z to be a convex combination of the entries of the vector c (and the coefficients in both convex combinations are the same).

Now consider the following geometric picture (which you can draw and visualize assuming that $m = 2$, where m is the number of rows in A). The geometric picture is drawn in R^{m+1}, where the last coordinate (the vertical one) is called the z direction.

There are vectors (points in R^{m+1}) y_i, one for each variable of the LP, where y_i is composed of column A_i and one extra coordinate with value c_i. There is a line in R^{m+1} parallel to the z direction that corresponds to all points that have b as their first m coordinates. We call it the b line.

A basic feasible solution is a set B of $m + 1$ affinely independent vectors from the y_i vectors, such that the b line intersects their convex hull (which is a simplex). The hyperplane on which this simplex lies is called the *dual plane*. This is the set of points that can be expressed as $\sum \lambda_i y_i$ for $y_i \in B$ and $\sum \lambda_i = 1$. The value of the objective function is the value of the z coordinate at the point of intersection.

During a pivot operation, we move from simplex to simplex, where the two simplices differ in one vertex.

The reduced costs have a geometrical interpretation in this picture (whose proof is left as homework). The reduced cost of variable x_j is exatly the distance one needs to travel from y_j in the z direction until one eventually intersects the dual plane. If y_j lies below this dual plane, then the reduced cost is negative.

A degeneracy is a place where the b line intersects a simplex of lower dimension (e.g., a line joining two points, when $m = 2$).

Remark: If the optimum value of the original LP is unbounded, then this is indicated by having $y = 0$ at the optimal solution for the new LP.