Notes by Richard C. Davis

- Linear Programming and Optimization
  1. Introduction
  2. Geometric Interpretation
  3. Simplex Method
  4. Improvements to Simplex

1 Introduction

Optimization is a very important problem in industry. Numerous companies use it to streamline their operations by finding optimal tradeoffs between demand for products or services, manufacturing costs, storage costs, repair costs, salary costs, etc.

- 50+ software packages available
- 1300+ papers just on interior-point methods
- 100+ books in the library
- 10+ courses at most universities
- Dozens of companies
- Delta Airlines claims they save $100 million a year with their optimization application

1.1 Some Optimization Problems

Optimization problems typically try to minimize (or maximize) some cost function, which is often called the *objective function* given some set of constraints. Here are some common optimization problems

Unconstrained Optimization:

$$\min\{f(x) : x \in \mathcal{R}^n\}$$

where f is an objective function of n real values.

Constrained Optimization:

$$\min\{f(x) : c_i(x) \leq 0, i \in \mathcal{I}, c_i(x) = 0, i \in \mathcal{E}, x \in \mathcal{R}^n\}$$

where $\mathcal{I}$ is a set of indices of inequalities, $\mathcal{E}$ is a set of indices of equalities, and c_i is a set of constraint functions each on n variables.

Quadratic Programming:

$$\min\{\frac{1}{2}x^T Qx + c^T x : a_i^T x \leq b_i, i \in \mathcal{I}, a_i^T x = b_i, i \in \mathcal{E} x \in \mathcal{R}^n\}$$

Here we constrain the objective function to be quadratic in the input variables x and the constraint functions to be linear.

Linear Programming:

$$\min\{c^T x : Ax \geq b, x \geq 0, x \in \mathcal{R}^n, c \in \mathcal{R}^n, b \in \mathcal{R}^m, A \in \mathcal{R}^{m \times n}\}$$

Here both the objective and the constraint functions are linear. We also do not include any equalities, but these are easy to express with the inequalities. We will see that there are many equivalent formats to express linear programs.

Integer Linear Programming:

$$\min\{c^T x : Ax \geq b, x \geq 0, x \in \mathcal{Z}^n, c \in \mathcal{R}^n, b \in \mathcal{R}^m, A \in \mathcal{R}^{m \times n}\}$$

Same as the linear program except that the variables x are integers instead of reals. Often referred to as just Integer Programs (IPs). We will also consider problems in which some of the variables are integers and some are reals. These are called *mixed integer linear programs* (MIPs).

In this class and the next we will be discussing solutions to linear programs. In the following two classes we will discuss integer programs. Note that while linear programs can be solved in polynomial time, solving integer programs is NP-hard. We will not discuss non-linear programs (general constrained optimization), although some of the solutions we will discuss for linear programs, *e.g.* the interior point methods, can also be applied to nonlinear objective functions.

1.2 Applications of Linear Programming

- As a substep in pretty much all the integer and mixed-integer linear programming (MIP) techniques.
- Selecting a mix: Oil mixtures, portfolio selection, ...
- Distribution: How much of a commodity should distribute to different locations.
- Allocation: How much of a resource should allocate to different tasks.
- Network Flows

Although linear programs do have a reasonable number of direct applications in industry, most larger applications require MIP programs since

1. many resources are integral, and
2. integer variables can be used to model yes/no decisions (0-1 programming).

Therefore the most used application of linear programming is probably the first one mentioned above (substep of MIP programs).

Name	Rows	Columns	Nonzeros
binpacking	42	415,953	3,526,288
distribution	24,377	46,592	2,139,096
forestry	1,973	60,859	2,111,658
maintenance	17,619	79,790	1,681,645
crew	4,089	121,871	602,491
airfleet	21,971	68,136	273,539
energy	27,145	31,053	268,153
4color	18,262	23,211	136,324

Table 9: Examples of Large Models

1.3 Overview of Algorithms

- Simplex (Dantzig 1947)
- Ellipsoid (Khachian 1979) - the "first" polynomial-time algorithm
- Interior Point - the "first" practical polynomial-time algorithm
  - Projective Method (Karmarkar 1984)
  - Affine Method (Dikin 1967)
  - Logarithmic Barrier Method (Frisch 1955, Fiacco 1968, Gill et. al 1986)

The word "first" appears in quotes because other polynomial-time algorithms were around when Khachian and Karmarkar published their work, but they were not known to be polynomial-time algorithms until after these algorithms were published. Note that many of the interior point methods can be applied to nonlinear programs.

In the constraint equation $Ax \geq b$ of a linear program, x and b are vectors and A is a matrix. Each row of A corresponds to one constraint equation and each column corresponds to one variable. In practice the matrix A is often sparse (most entries are zero) since each constraint typically only depends on a few of the variables. This allows many of the algorithms to use special, fast subroutines designed to work with sparse-matrices. The current state of the art in such solvers allows us to work with problems where the matrices have 10-100 thousand rows/columns, and where 100 thousand to 1 million of the items in the matrices are nonzeros. Table 9 shows the size of some problems.

Though the simplex algorithm is not a polynomial-time algorithm in the general case (contrived examples can take exponential time), it is very fast for most problems. For this reason, it is still an open question as to whether or not interior point methods are "superior" to the simplex method and for what problem types. Here are two quotes that illustrate the battle between simplex and interior point methods:

- "The results of this paper demonstrate that the initial claims of the computational superiority of interior point methods are now beginning to appear." - Lustig, Marsten and Shanno, 1994 (ORSA Journal of Computing, 6 (1)).

Name	Simplex (Primal)	Simplex (Dual)	Barrier + Crossover
binpacking	29.5	62.8	560.6
distribution	18,568.0	won't run	12,495,464
forestry	1,354.2	1,911.4	2,348.0
maintenance	57,916.3	89,890.9	3,240.8
crew	7,182.6	16,172.2	1,264.2
airfleet	71,292.5	108,015.0	37,627.3
energy	3,091.1	1,943.8	858.0
4color	45,870.2	won't run	44,899,242

Table 10: Running Times of Large Models in seconds.

- "It is our opinion that the results of Lustig, Marsten and Shanno somewhat overstate the performance of [the interior point methods] relative to the simplex method." - Bixby, 1994 (ORSA Journal of Computing, 6 (1)).

Table 10 shows a comparison of the simplex method (running on either the original problem, or the *dual*, as explained in the next lecture) and an interior point method, the Barrier + Crossover method. The example problems are the same ones in Table 9. Note that the algorithms have widely varying performance depending on the problem given. The running time of both algorithms is very much dependent on the type of problem. This is part of why it is difficult to see whether or not interior point methods are "superior" to the simplex method.

It is clear, however, that (1) interior point methods are becoming more dominant over time and (2) the appearance of interior point methods has sparked a resurgence of interest in finding more efficient techniques for the simplex method.

1.4 Linear Programming Formulation

There are many ways to formulate linear programs, and different algorithms often use different formulations as inputs. Fortunately it is typically easy to convert among the formulations. For example the formulation presented earlier

minimize $c^T x$ (this is the *objective* function)
subject to $Ax \geq b$, $x \geq 0$

can be converted to

minimize $c^T x$
subject to $Ax = b$, $x \geq 0$

by adding *slack* variables. For example:

$$7x_1 + 5x_2 \geq 7$$

can be converted to

$$7x_1 + 5x_2 - y_1 = 7 \text{ and } y_1 \geq 0$$

That is, slack variables like y_1 must be added to the vector x for each row or A to convert the expression $Ax \geq b$ to $Ax = b$. This formulation with the equality constraints $Ax = b$ is the formulation that is used by the simplex method.

1.5 Using Linear Programming for Maximum-Flow

Before describing the algorithms for solving linear programs we describe how linear programs can be use to solve maximum flow problems. Consider the flow problem shown in Figure 36. In this problem the numbers on the edges represent the maximum flow allowed through the edges (its capacity). The goal is to maximize the total flow from the input (IN) to the output (OUT) with the constraints that the net flow into each node has to be zero, and no flow along an edge can be greater than the capacity. To convert this to a linear programming problem, take every edge e_i and create two edges, x_i and x'_i, one for positive flow in one direction, and the other for positive flow in the other direction. The linear programming formulation will have one variable for each direction of each edge.

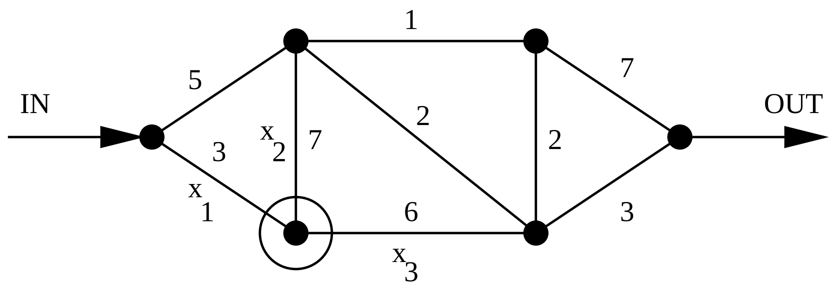

Figure 36: A network maximum-flow problem.

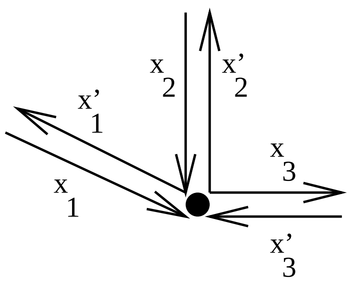

Figure 37: Part of a Network Converted for Linear Programming

For each vertex, we add a constraint that forces the flow into the vertex to equal the flow out of the vertex. Look at the highlighted vertex in Figure 36 connected to the edges labeled x_1, x_2, and x_3. This will be converted to the network shown in Figure 37. Adding the constraint that input flow equal output flow, we have

$$x_1 + x_2 + x'_3 = x'_1 + x'_2 + x_3$$

We then add two inequality constraints per edge to include the capacity constraints. For edge x_1 in Figure 36, these inequalities will be $x_1 \leq 3$ and $x_1' \leq 3$. Finally, we add one more edge x_0 connecting "OUT" to "IN", but we place no constraints on this flow. This completes the formulation of the network flow problem as a linear programming problem. Maximizing the value of x_0 will find the maximal flow for this network.

2 Geometric Interpretation

We can visualize linear programming problems geometrically. We present a two dimensional example. (Here we are maximizing instead of minimizing, but the ideas are still the same.)

$$\begin{aligned}
\text{maximize } z &= 2x_1 + 3x_2 \\
\text{subject to } x_1 - 2x_2 &\leq 4 \\
2x_1 + x_2 &\leq 18 \\
x_2 &\leq 10 \\
x_1, x_2 &\geq 0
\end{aligned}$$

Each inequality represents a halfspace, as shown in Figure 38. The intersection of these halfspaces is the region of points that abide by all the constraints. This is called the *feasible region*. The figure also shows that the objective function increases in one direction. The maximal value of x is the point in the feasible region that is farthest in the direction of this vector.

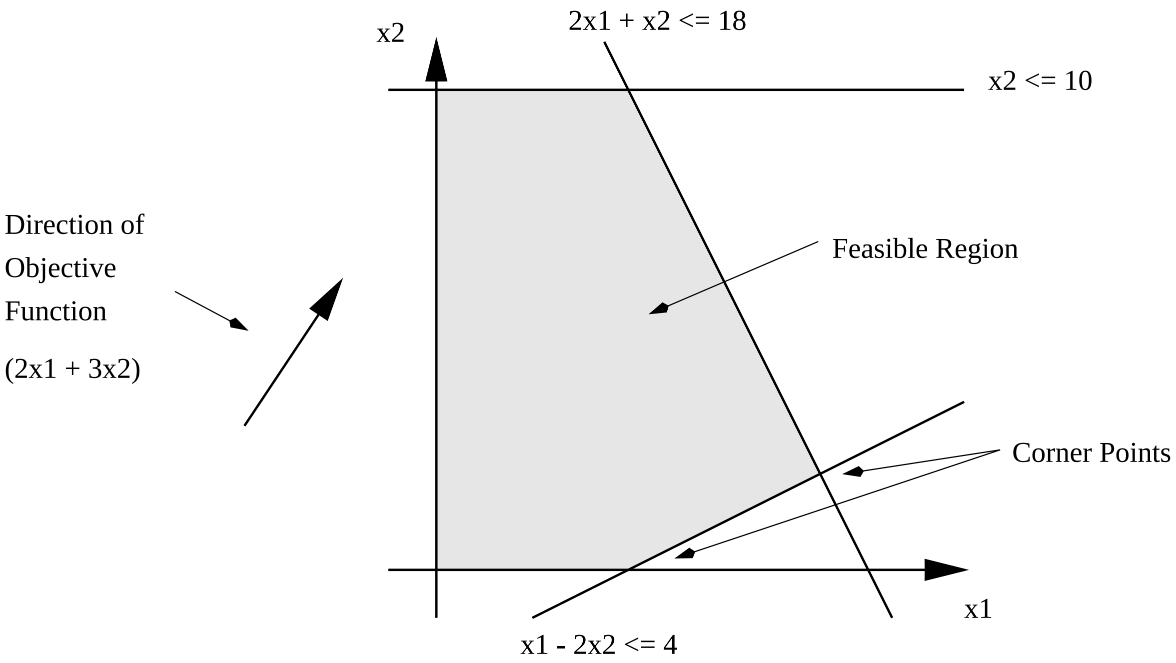

Figure 38: Geometric interpretation of Linear Programming

There are some interesting things to note here. First of all, the maximal point *must* be on a *corner* formed by the intersection of the halfspace boundaries. These corners are sometimes called *extreme points*. Also note that the feasible region could be infinite, allowing the maximum value of the objective function to be infinite, or it might not exist at all, meaning there is no solution.

As discussed earlier, we can add the slack variables x_3, x_4, and x_5, to the first, second, and third constraints, and the new formulation will look as follows.

$$
\begin{aligned}
\text{maximize } z &= 2x_1 + 3x_2 \\
\text{subject to } x_1 - 2x_2 + x_3 &= 4 \\
2x_1 + x_2 + x_4 &= 18 \\
x_2 + x_5 &= 10 \\
x_1, x_2, x_3, x_4, x_5 &\geq 0
\end{aligned}
$$

Figure 39 shows a geometric interpretation for slack variables. As each variable increases, we move further into the feasible region. When the slack variable is 0, we are actually on the line corresponding to that slack variable.

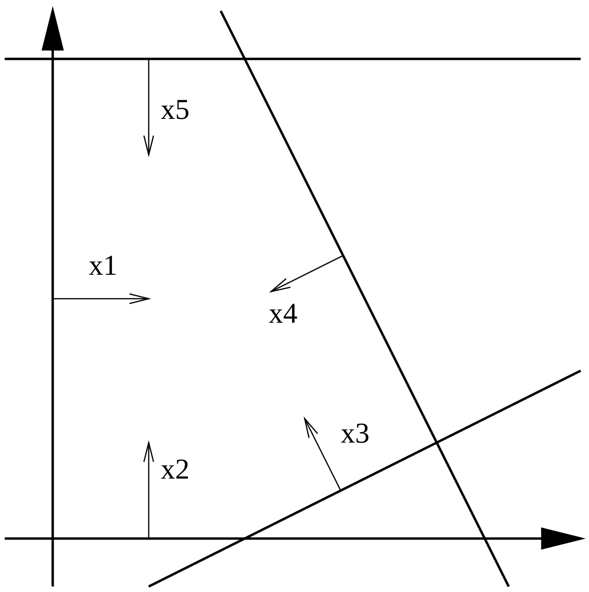

Figure 39: Geometric interpretation of Slack Variables

3 Simplex Method

This is one of the earliest methods for solving linear programs. Its running time is exponential in the worst case, but for many problems it runs much faster than that. Intuitively, the basic algorithm works as follows

1. Find any corner of the feasible region (if one exists).

2. Let's say this corner is at the intersection of n hyperplanes. For each plane, calculate the dot product of the objective function cost vector c with the unit vector normal to the plane (facing inward).

3. If all dot products are negative, then DONE (the problem is already maximized).
4. Else, select the plane with the maximum dot product.
5. The intersection of the remaining $n-1$ hyperplanes forms a line. Move along this line until the next corner is reached.
6. Goto 2.

Following these steps, the algorithm moves from corner to corner until the maximal solution is reached. At each step, it moves away from a plane in such that the rate of change of the objective function is maximized. There are actually different possible ways of deciding which plane to leave and more complicated variants of simplex use different methods. Figure 40 shows how simplex can start at (0,0) and move to the maximal solution. On the first step it chooses the hyperplane (line in 2 dimensions) marked $x2$ to move away from, and on the second step it chooses the hyperplane marked $x1$.

At each step, the variable corresponding to the hyperplane that the current point leaves is called the *entering* variable, and the variable corresponding to the hyperplane that the current point moves to is called the *departing* variable. In the first step of the example in Figure 40 variable $x2$ is the entering variable and variable $x5$ is the departing variable. This sounds counter-intuitive, but it will make more sense as we delve into the details of simplex. In particular the value of an entering variable will change from zero to nonzero, while the value of a departing variable will change from nonzero to zero.

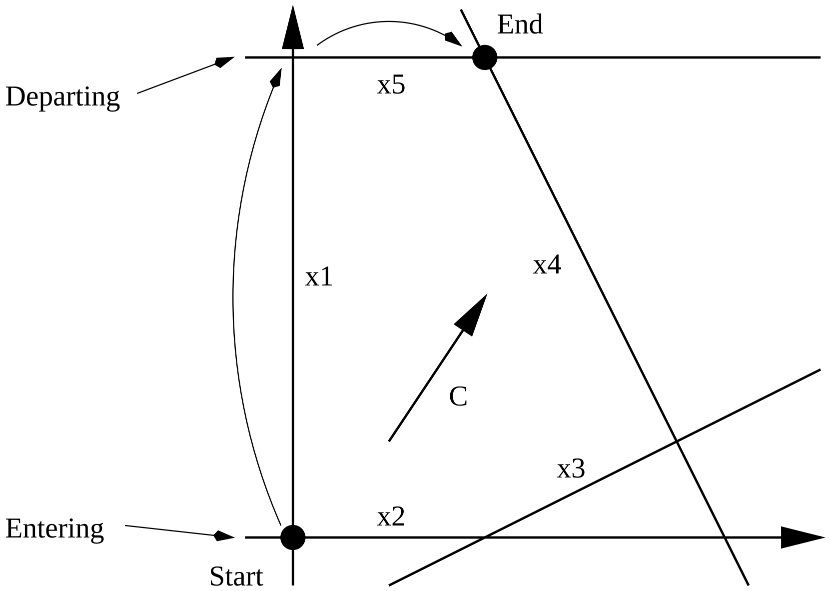

Figure 40: Geometric View of the Simplex Algorithm

Note that this is a geometrical explanation of simplex. The actual algorithm is a little cleverer than this... we don't have to find a unit vector pointing in the direction of each of the outgoing lines at each step.

3.1 Tableau Method

The Tableau Method is a rather elegant organization for the simplex method. Our formulation once again is as follows.

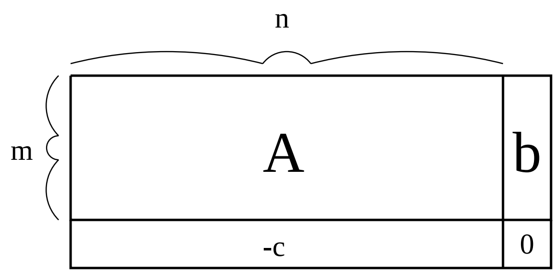

Figure 41: The Tableau

maximize $c^T x$
subject to $Ax = b$, $x \geq 0$

The Tableau for this problem is shown in Figure 41. It is an $(m+1)\times(n+1)$ matrix. The $m+1$ rows correspond to the m constraint equations plus the negative of the cost vector. (Without going into too much detail, it is negative because we are maximizing. If we were minimizing, the vector c would be positive here.) The $n+1$ columns correspond to the n components of the vector x (including original variables and slack variables) and the vector b in the equation $Ax = b$.

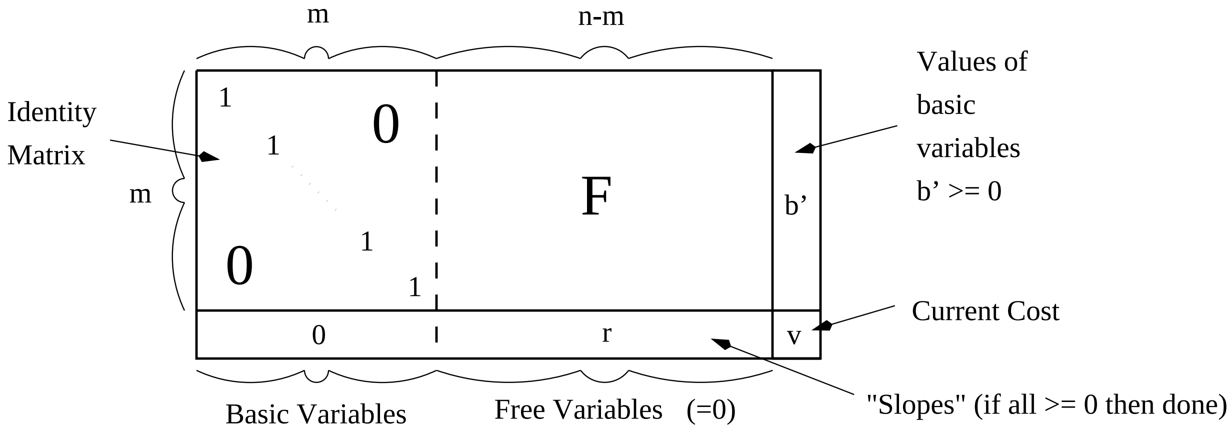

Figure 42: The Tableau after an Initial Point has been Found

The first step of the simplex algorithm is to find a corner (extreme point) of the feasible region. For the moment, let's forget about how we find this point, and look a what this does to the Tableau, as shown in Figure 42.

In a corner, $n-m$ of the variables must have the value 0 (corresponding to the intersection of $n-m$ hyperplanes). One way to see why this happens is to refer to Figure 39. Since, we have 5 variables, and 3 constraints, 2 variables must be 0. Remember that when a variable has a zero value, the current point lies on that hyperplane. Since a corner, in this example, is where two lines meet, two of our variables must be 0. This is an easy way to see what is happening in a tableau. We'll return to this example later.

The $n-m$ variables that are 0 are the *free variables*. The other m variables have positive values, and are called the *basic variables*. If we arrange our tableau so that the m basic variables are on the left, we can use Gaussian elimination on the tableau to put it in the form shown in Figure 42, with ones on the diagonal. When arranged this way, the elements of the vector b' on the right become the values of the basic variables at the current point x (remember, the free variables are all 0). Furthermore, since we are assuming the solution is feasible then all the elements of b' must be nonnegative.

We can also use Gaussian elimination to make components of the cost vector $-c$ corresponding to the basic variables equal to 0. Through the process of elimination, the bottom, right corner of the tableau comes to hold the current cost. Also, the components of the cost vector corresponding to the free variables become the *slopes*, i.e. the rate at which the current cost would decrease if that free variable were increased from 0.

Now we can see how the simplex method proceeds. At each step we are given a tableau like the one in Figure 42. The most negative element of the cost vector tells us which free variable we need to increase from zero. In our two-dimensional example of Figure 39, this tells us which line we need to move away from in order to maximally increase our objective function. The variable x_i which we must increase is called the *entering variable* because it is entering the set of basic variables. Figure 43 (a) illustrates this step.

We have chosen which line to move away from, but we have not chosen which line to move to. In other words, we have to choose a *departing variable* which will leave the set of basic variables and become zero. Figure 43 (b) shows how this is done. Look at the vector u corresponding to the entering variable chosen above, and look at the vector b' that contains the values of the basic variables at our current point x. For each basic variable x_i, compute the ratio b'_i/u_i. This ratio indicates how much the objective function can increase if we use x_i as the departing variable. In the 2-D example, this tells us how much the function can increase if we move from one line to the other. (Note: it cannot increase more than this, because we would have to leave the feasible region). We must choose the basic variable with the smallest ratio as the departing variable, because we cannot leave the feasible region. Note: this ratio is taken only over the positive components of u, because a negative ratio would indicate that the cost function would *decrease* if we moved toward that line, and we don't want to do that!

Now that the entering variable and the departing variable have been chosen, we simply swap their columns in the tableau, as shown in Figure 43(c), and then perform Gaussian elimination, as shown in Figure 43 (d), to get the tableau back into the state it was at the beginning of the simplex step. We repeat these steps until there are no negative slopes (in which case there is no way to increase the objective function and the maximum has been reached), or until there are no positive ratios b'_i/u_i (in which case the maximum value is infinite - it is difficult to explain why this is true, but it is).

An example is in order. Figure 44 shows how the tableau would look for our earlier example in Figures 38 and 39. Figure 44 (a) shows the initial tableau for this example. Our initial point is (0,0), so it only takes a little rearranging to put the tableau in the state

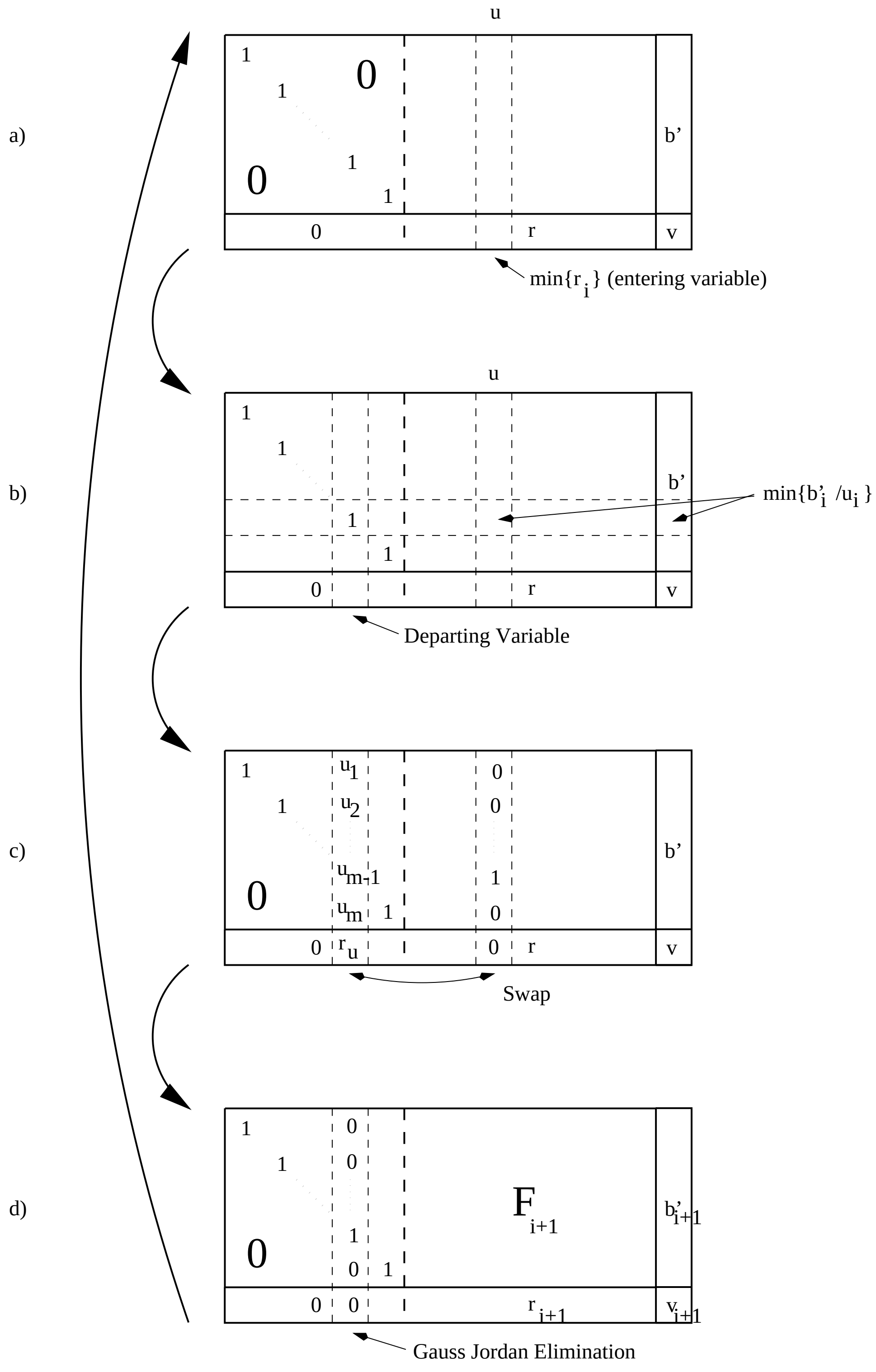

Figure 43: Simplex Steps Performed on a Tableau

1	-2	1	0	0	4
2	1	0	1	0	18
0	1	0	0	1	10
-2	-3	0	0	0	0
x1	x2	x3	x4	x5	

x1-2x2+x3 = 4

2x1+x2+x4 = 18

x2+x5 = 10

2x1+3x2

Initial Tableau
(a)

1	0	0	1	-2	4
0	1	0	2	1	18
0	0	1	0	1	10
0	0	0	-2	-3	0
x3	x4	x5	x1	x2	

x1 = x2 = 0
Starting Point Chosen
(b)

				u	
1	0	0	1	-2	4
0	1	0	2	1	18
0	0	1	0	1	10
0	0	0	-2	-3	0
x3	x4	x5	x1	x2	

Choose Minimum Value
For Maximum Slope
(c)

						bi/ui
1	0	0	1	-2	4	-2
0	1	0	2	1	18	18
0	0	1	0	1	10	10
0	0	0	-2	-3	0	
x3	x4	x5	x1	x2		

min

swap

Find Minimum Positive Ratio
(d)

1	0	-2	1	0	4
0	1	1	2	0	18
0	0	1	0	1	10
0	0	-3	-2	0	0
x3	x4	x2	x1	x5	

Perform Swap
(e)

1	0	0	1	2	24
0	1	0	2	-1	8
0	0	1	0	1	10
0	0	0	-2	3	30
x3	x4	x2	x1	x5	

Perform Gaussian Elimination
to Complete the Step
(f)

Figure 44: An Example of Simplex Steps on a Tableau

shown in Figure 44 (b). In Figure 44 (c) we find the minimum value of the cost vector, which corresponds to the maximal slope. Figure 44 (d) shows the ratios b'_i/u_i, and which columns we choose to swap. In Figure 44 (e) we see the tableau after the swap, and in Figure 44 (f) we see the tableau after Gaussian elimination.

3.2 Finding an Initial Corner Point

We have not yet discussed how to find the initial corner point needed to start the simplex method. If we were originally given the problem in the form $Ax \leq b, x \geq 0$ and $b \geq 0$, then we can add slack variables and use them as our initial basic variables (*i.e.*, our initial corner is where all variables x are 0 and the slack variables y equal b). However, if we are given the problem in the form $Ax = b$, we will need to find an initial corner. One way to do this is to use the simplex method to solve a slightly modified version of the original problem. Given the tableau in Figure 41, we can extend it further by adding m artificial variables, one for each row (these are analogous to the m slack variables we previously considered). These will form an identity matrix as shown in Figure 45. (Note: the vector b must be positive, but any problem can be put in this form by negating the rows with negative values in b.)

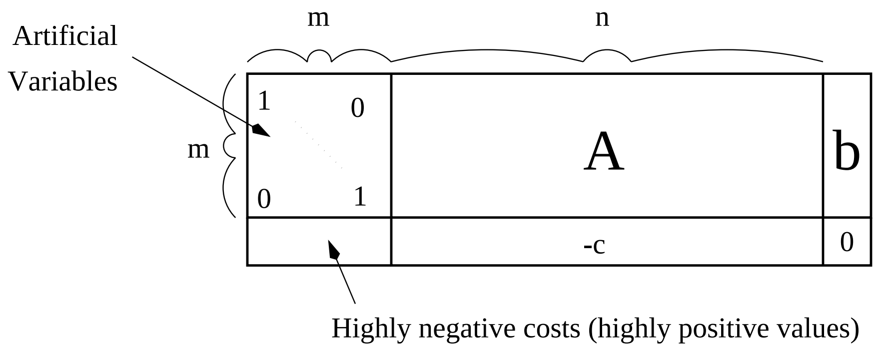

Figure 45: A Modified Tableau that will find a Starting Corner Point

If we give each of these artificial variables a highly negative cost (which corresponds to high positive values in the tableau because it holds $-c$). We can solve this with the simplex method as if it were a normal tableau. The artificial variables will be the set of basic variables in the first step (again, we can make the cost vector 0 below them by Gaussian elimination) and the other variables will be free.

Our vector x will quickly move away from the artificial variables because the algorithm maximizes the objective function (cost) and these variables have high negative values. Soon, the artificial variables will be free variables, and some other set of m variables will be the basic variables. When this happens, we can remove the columns corresponding to the artificial variables, and we will have a tableau that we can use to start solving the original problem. On the other hand, if this process ends and there are still artificial variables in the basis, then there is no feasible region for the original problem.

3.3 Another View of the Tableau

Figure 46 shows another way to look at the tableau. Once we have moved the basic variables to the left and the free variables to the right, we end up with the matrix shown in Figure 46 a. Gaussian elimination turns the matrix B into the identity matrix, which is equivalent to left-multiplying the top three blocks by B^{-1}. Elimination of c_b on the bottom row is equivalent to subtracting c_b times the top of the matrix from the bottom. This is shown in Figure 46 b.

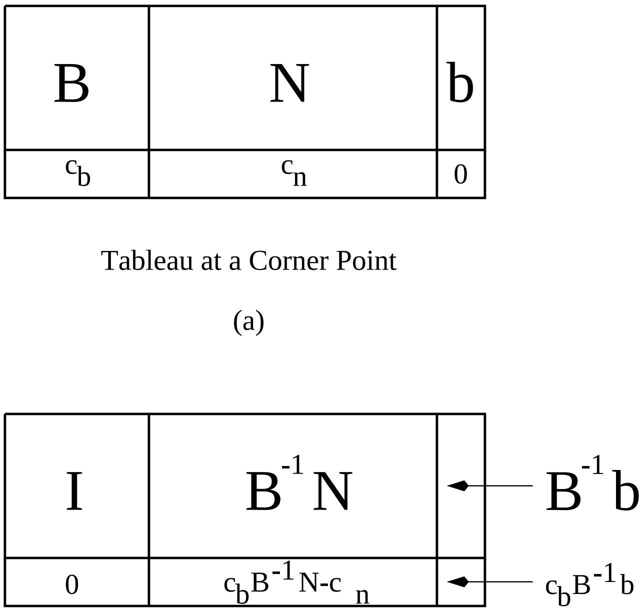

Tableau at a Corner Point

(a)

Tableau after Gaussian Elimination

(b)

Figure 46: Another way to look at a Tableau

Note that the tableau is likely to be very dense. Even when the original formulation of the problem is very sparse the inverse B^{-1} will typically be dense.

3.4 Improvements to Simplex

The density of the tableau, mentioned in the previous section, is a major problem. Typically the matrix A starts out very sparse—it is not unusual that 1 out of every 10,000 elements that are nonzero. Since B^{-1} will be dense, however, algorithms cannot afford to calculate it directly. For this reason, it is common to keep the matrix B^{-1} in the factored form...

$$U_1^{-1} U_2^{-1} \cdots U_m^{-1} L_m^{-1} \cdots L_1^{-1}$$

...which can be much sparser than B^{-1}. The trick (and major improvement over the last decade) is updating this factorization in place when exchanging variables.

Here are some other improvements.

- Improved pricing (i.e. don't always go down the maximum slope). One example of this is *normalized pricing* in which prices are weighted by the length of the path to the departing variable.
- Better factoring routines (for initial factorization).
- "Perturbation Method" for dealing with stalling. The Simplex method can stall when it hits a degeneracy (i.e. a place where more than m lines meet in a corner in the 2-D case - one of them is redundant) and will sometimes stall by leaving a redundant line, only to return to the same corner.
- Various hacks in how to store the sparse matrices (*e.g.* store both in column and row major order).
- Better Processing.

Speeds have improved by 3-100 times (depending on model) due to algorithmic improvements over the last 10 years.