Linear Programming

April 12, 2005

Parts of this were adapted from Chapter 29 of Introduction to Algorithms (Second Edition) by Cormen, Leiserson, Rivest and Stein.

1 What is linear programming?

The first thing to know about linear programming is that it is programming in the sense of trying to establish a plan or a schedule for something rather than programming in the sense of trying to write some code to solve a problem. For example, a TV station has to solve the problem of deciding which shows to air at which times. This is a programming problem.

The kind of programming we are going to study is linear. Linear means that the constraints of the cost or profit of a decision are expressed as linear combinations of variables. A linear combination is a sum of constant-coefficients times variables that is less than, greater than or equal to a constant. The variables are raised to the first power.

In most cases, a linear programming problem will look something like this:

Maximize
\[ x_1 - 5x_2 + 3x_3 + 3 \]
Subject to
\[ x_1 - x_2 > 3 \]
\[ x - 2 + 2x_3 < 2 \]
\[ 5x_1 - 2x_2 + x_3 > 7 \]

The first equation is the objective function (which term should be familiar to you from greedy algorithms) and the other equations are the constraints. Another way to think about linear programming is as a constraint satisfaction and optimization problem.

It is not hard to make up simple examples of linear programming problems based on business problems. For example, suppose a furniture production company makes tables, chairs and window frames. The company can get a 50 dollar profit from selling a table, a 10 dollar profit from selling a chair and a 15 dollar profit from selling a window frame. The production of furniture requires paying a cost for supplies and labor. The problem here is to decide how many tables, chairs and window frames to make given the cost of each item. More
concretely, if we let $t =$ the number of tables produced, $c =$ the number of chairs produced, $w =$ the number of window frames produced, $s =$ the cost of supplies and $l =$ the cost of labor then we want to maximize

$$50t + 10c + 15w - s - l$$

subject to a set of equations that describe the cost of producing each item and a equations that describe how much money we have on hand for buying labor and supplies. Suppose a table costs 20 dollars in supplies and 30 dollars in labor then we would add the constraints

$$30t + l = 0$$
$$20t + s = 0$$

which require us to spend 30 dollars on labor for each table built and 20 dollars on supplies for each table built. Similarly, suppose it costs 10 dollars in supplies and 10 dollars in labor to make a chair and that it costs 15 dollars in supplies and 5 dollars in labor to make a window frame. We would add the following constraints

$$10c + l = 0$$
$$10c + s = 0$$

and we would have a problem right there. The problem is that the cost of building the tables and the cost of building the chairs are entangled. So we need different cost variables for each item. We go back and add them in to the objective function and the constraint equations to get

Maximize
$$50t + 10c + 15w - s_t - l_t - s_c - l_c - s_w - l_w$$
Subject to
$$30t + l_t = 0$$
$$20t + s_t = 0$$
$$30c + l_c = 0$$
$$20c + s_c = 0$$
$$30w + l_w = 0$$
$$20w + s_w = 0$$

The final constraint is that we only have 400 dollars on hand to spend on labor and supplies. That constraint is easily expressed as

$$l_t + s_t + l_c + s_c + l_w + s_w \leq 400$$

We can now pass our linear programming problem off to our favorite solver and find how the most profitable way to program our furniture shop so as to maximize profits.

In the furniture problems, we tacitly assumed that $t, c$ and $w$ were integers. That is, we assumed we could only make an integer number of tables, chairs and window frames. This assumption makes sense for this problem. But for most problems we will assume that all variables can have arbitrary real values.
One interesting thing about linear programming is that, compared to graph formulations of problems, it is an entirely different way to think about a problem. In the graph representation, the challenge is to define the contents of each node in the graph, describe how child nodes are related to their parents and describe an algorithm for exploring the graph. In linear programming, the challenge is to formulate the problem as a set of equations and then solve them.

2 Solving Linear Programming Problems

In this section, we will learn how to solve linear programming problems using the simplex method. The simplex method is a greedy algorithm, which brings us full circle in algorithm design. You may recall that we studied greedy algorithms first in 312. Interestingly, we can formulate the continuous knapsack problem as a linear programming problem and use the simplex method to derive an algorithm that looks a lot, very exactly, like the greedy knapsack algorithm. But we digress. First, we need some standardized forms for linear programming problems. This will simplify our discussion.

2.1 Standard Form

The standard form for a linear program is

Maximize \( \sum_{j=1}^{n} c_j x_j \)

subject to

\( \sum_{j=1}^{n} a_{ij} x_j \leq b_i \) for \( i = 1, 2, \ldots m \)

\( x_j \geq 0 \) for \( j = 1, 2, \ldots n \)

in which all variables range over the real numbers. Given a linear programming problem, it is always possible to add variables and constraints to get the problem into standard form. We will assume all problems are in standard form henceforth.

The standard form includes the optimization of the objective function with \( n \) variables \((x_1 \ldots x_n)\) using \( m \) constraint equations of \( n \) terms each and a set of \( n \) nonnegativity equations. It is often convenient (especially when implementing a linear program solver) to represent a standard form linear programming problem as a pair of vectors. The \( m \times n \) matrix \( A \) contains the values of the \( a \) coefficients in the constraint equations, an \( n \)-dimensional vector \( b \) contains the \( b_i \) entries from the constraint equations and an \( n \)-dimensional vector \( c \) contains the \( c_i \) entries from the objective function.

If we use the matrix and vectors representation, then the standard form is

Maximize \( c^T x \)

subject to

\( Ax \leq b \)

\( x \geq 0 \)
in which $c^T$ is the transposition of $c$ (basically you just rotate $c$ 90 degrees and multiply a 1 by $n$ matrix times an $n$ by 1 matrix to get a single scalar value) and 0 is the $n$-dimensional 0 vector.

### 2.2 Slack Form

Standard form is nice because it means that we can assume that all linear programming problems are in the same form. That simplifies the definition of slack form. We care about slack form because that is the form used in the simplex method. In slack form, the only inequalities allowed are nonnegativity constraints.

Converting to slack form requires two changes. First, a new variable, $z$ is introduced to store the value of the objective function. Second, all of the constraint equations from standard form are rewritten as equalities, rather than inequalities, with a new nonnegativity constraint. So the constraint

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i$$

becomes the pair of constraints

$$s = b_i - \sum_{j=1}^{n} a_{ij} x_j$$

$$s \geq 0 .$$

The important thing here is that the pair of constraints is satisfied if and only if the original constraint is satisfied. The variable $s$ is called a *slack variable* because it measures the difference, or slack, between the left and right sides of the original constraint.

It is convenient to assume that the slack variable for constraint $i$ is called $x_{n+i}$. So that the slack form is

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j$$

$$x_{n+i} \geq 0 .$$

Another name for the slack variables plus $z$, the objective function value, is basic variables. The other $x_i$ variables are called non-basic variables.

At this point, it’s probably a good idea to do an example. This example comes from the Cormen, Leiserson, Rivest and Stein book. We will begin with the linear programing problem

Maximize $2x_1 - 3x_2 + 3x_3$

subject to

$$x_1 + x_2 - x_3 \leq 7$$

$$-x_1 - x_2 + x_3 \leq -7$$

$$x_1 - 2x_2 + 2x_3 \leq 4$$

$$x_1, x_2, x_3 \geq 0$$
The first thing to notice is that this linear programming problem is indeed in standard form. We want to convert it to slack form. First, we will add the slack variables $x_4, x_5, x_6$ and change the constraints to equalities.

Maximize $2x_1 - 3x_2 + 3x_3$

subject to

$x_4 = 7 - x_1 - x_2 + x_3$
$x_5 = -7 + x_1 + x_2 - x_3$
$x_6 = 4 - x_1 + 2x_2 - 2x_3$

$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$

Finally, we add the $z$ variable to track the value of the objective function.

$z = 2x_1 - 3x_2 + 3x_3$
$x_4 = 7 - x_1 - x_2 + x_3$
$x_5 = -7 + x_1 + x_2 - x_3$
$x_6 = 4 - x_1 + 2x_2 - 2x_3$

$x_1, x_2, x_3, x_4, x_5, x_6, z \geq 0$

Now we want a more concise representation of a linear programming problem in slack form. Just like before, we will use the matrix $A$ and the vectors $b$ and $c$ to keep track of the coefficients in the contraints (that’s $A$), the constant terms in the slack equations (that’s $b$) and the coefficients in the objective function (that’s $c$). But, we will also need to know the indices for the basic variables and the indices for the non-basic variables. The set $B$ will contain the indices of the basic variables and the set $N$ will contain the indices of the non-basic variables. Finally, the value $v$ will be an optional constant term in the objective function.

The compact representation for the above linear program in slack form is:

$B = \{4, 5, 6\}$
$N = \{1, 2, 3\}$
$c = \begin{pmatrix} 2 & -3 & 3 \end{pmatrix}^T$
$b = \begin{pmatrix} 7 \\ -7 \\ 4 \end{pmatrix}$
$A = \begin{pmatrix} 1 & 1 & -1 \\ -1 & -1 & 1 \\ 1 & -2 & 2 \end{pmatrix}$
$v = 0$

Notice that the coefficients of $A$ are the negative of how they appear in slack form. The variable $v$ is an optional constant term in the objective function. In this example, there is not constant term in the objective function so it is set to 0.
3 Simplex Algorithm

The simplex algorithm is similar to how a linear system is solved using Gaussian elimination. In Gaussian elimination, the system is repeatedly transformed into an equivalent system until its structure is such that a solution can be easily extracted. The simplex algorithm iterates in a similar manner. We associate with each iteration of the algorithm a basic solution. A basic solution is obtained from the slack form of the linear programming by setting each nonbasic variable to 0 and then compute the values of the basic variable from the equality constraints. A basic solution corresponds to a vertex of the simplex. On each iteration of the algorithm, we are going to convert the current slack form into a new equivalent slack form whose basic solution is a different vertex of the simplex. The goal is to move through the vertexes to find the one to maximize our object value. This is done by finding a nonbasic variable that when increased from 0 causes an increase in the objective value. The amount by which we can increase the nonbasic variable is limited by the basic variables. In particular, we increase the nonbasic variable until one of the basic variables become zero. This indicates that there is no more slack in the system (i.e., the difference between the function and its constraining inequality is 0). At this point we rewrite the slack form to exchange the chosen nonbasic variable with the basic variable that is now 0. This is a new basic solution and we repeat the process until we cannot find a nonbasic variable to increase that gives an increase in the objective function.

3.1 Simplex Algorithm Example

An extended example will help see the process. This example is adapted from Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein. Consider the following linear program in standard form:

maximize \[ 3x_1 + x_2 + 2x_3 \]
subject to \[ \begin{align*}
x_1 + x_2 + 3x_3 & \leq 30 \\
2x_1 + 2x_2 + 5x_3 & \leq 24 \\
4x_1 + x_2 + 2x_3 & \leq 36 \\
x_1, x_2, x_3 & \geq 0
\end{align*} \]

We first begin by converting this into its equivalent stack form. Slack form is useful for algebraic as well as algorithmic manipulation. We say that a constraint is tight for a particular setting of its nonbasic variables if they cause the constraint’s basic variable to become 0. Similarly, a setting of the nonbasic variables that would make a basic variable become negative violates that constraint. The slack variable thus represent how far away a constraint is from being tight, and they help us determine the limit to which we can increase nonbasic variables without violating any constraints.

We convert to slack form by creating a slack variable for each constraint, converting the inequality to equality constraints, and creating a variable for the
objective function. Note the we drop the nonegativity constraints since they are implied by the slack form. The slack form of the problem is:

\[
\begin{align*}
  z &= 3x_1 + x_2 + 2x_3 \\
  x_4 &= 30 - x_1 - x_2 - 3x_3 \\
  x_5 &= 24 - 2x_1 - 2x_2 - 5x_3 \\
  x_6 &= 36 - 4x_1 - x_2 - 2x_3
\end{align*}
\]

A feasible solution to this system is any positive setting of \( x_1, x_2, \) and \( x_3 \) that yields positive values for \( x_4, x_5, \) and \( x_6; \) thus, there are an infinite number of feasible solutions to this system. We are interested in the basic solution obtained by setting all the nonbasic variables to zero (i.e., the variables on the right-hand side of the equalities). The basic solution for our example is \((\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5, \bar{x}_6) = (0, 0, 0, 30, 24, 36)\), and it has an objective value \( z = (3/0) + (1 \cdot 0) + (2 \cdot 0) = 0 \). Note that the basic solution set \( \bar{x}_i = b_i \) for each \( i \in B \). An iteration of the simplex algorithm rewrites the objective function and constraint equations to create a different set of nonbasic variables. This in turn creates a new basic solution. The rewrite in no way changes the underlying linear program that is being solved. Rather, it represents the movement from one vertex to another in exploring the simplex. Please note that it is possible that the basic solution is not feasible in the first few iterations of the algorithm. This is OK.

We reformulate the linear programming problem in an iteration of the algorithm by picking a nonbasic variable \( x_e \) and increasing its value to see if it increases the objective function value. If this is the case, then we continue to increase \( x_e \) until a basic variable \( x_l \) becomes 0. At this point, \( x_e \) becomes basic (i.e., it has a nonzero value) and \( x_l \) becomes nonbasic (i.e., it has a zero value). We choose our nonbasic value to increase by looking at the objective function. If a nonbasic variable exists in the objective function with a positive coefficient, then we can select that variable as \( x_e \) in our algorithm. Keep in mind that as we increase \( x_e \) other basic variables and our objective functions may also change. The first basic variable to reach 0 becomes \( x_l \).

To continue with our example, notice that \( x_1 \) is a nonbasic variable with a positive coefficient in the objective function. We select this as the variable to increase. As we increase \( x_1 \), the variables \( x_4, x_5, \) and \( x_6 \) decrease. The nonnegativity constraint prevents us from increasing \( x_1 \) above 9 since \( x_6 \) becomes negative at that point; thus, \( x_6 \) is our tightest constraint since it limits how much we can increase \( x_1 \). We know switch the roles of \( x_1 \) and \( x_6 \) by solving the third constraint for \( x_6 \):

\[
\begin{align*}
  x_6 &= 36 - 4x_1 - x_2 - 2x_3 \\
  4x_1 &= 36 - x_2 - 2x_3 - x_6 \\
  x_1 &= 9 - \frac{x_2}{4} - \frac{x_3}{4} - \frac{x_6}{4}
\end{align*}
\]

We now rewrite the other equations by writing \( x_1 \) in terms of \( x_2, x_3, \) and \( x_6 \) using the above equation. Doing this for \( x_4 \) give us

\[
\begin{align*}
  x_4 &= 30 - x_1 - x_2 - 3x_3
\end{align*}
\]
\[
\begin{align*}
= & \ 30 - \left( 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4} \right) - x_2 - 3x_3 \\
= & \ 21 - \frac{3x_2}{4} - \frac{5x_3}{2} + \frac{x_6}{4}
\end{align*}
\]

We repeat this procedure for the remaining constraint and objective function to rewrite our linear program in the following form:

\[
\begin{align*}
z &= 27 + \frac{x_2}{4} + \frac{x_3}{2} - \frac{3x_6}{4} \\
x_1 &= 9 - \frac{x_2}{4} - \frac{2x_3}{2} - \frac{2x_6}{4} \\
x_4 &= 21 - \frac{3x_2}{4} - \frac{5x_3}{2} + \frac{x_6}{4} \\
x_5 &= 6 - \frac{3x_2}{2} - 4x_3 + \frac{x_6}{2}
\end{align*}
\]

The rewrite operation as shown above is called a **pivot**. A pivot takes the nonbasic variable \(x_e\) called the *entering variable* and the basic variable \(x_l\) called the *leaving variable* and exchanges their roles in the linear program.

Pivoting rewrites that linear program into an equivalent form. The original basic solution to our linear program was \((0,0,0,30,24,36)\) with an objective value of 0. The new basic solution to the linear program after the pivot is \((9,0,0,21,6,0)\) with an objective value of 27. This solution is feasible in our linear program before the pivot, and it yields the same objective value.

Continuing the example, we now find a new nonbasic variable to increase. We do not want to increase \(x_6\) since it has a negative coefficient. We can attempt to increase either \(x_2\) or \(x_3\). Let us use \(x_3\) We can increase \(x_3\) to \(\frac{3}{2}\) before the third constraint becomes negative, so the third constraint is the tightest constraint. We pivot on \(x_3\) and \(x_5\) by solving for \(x_3\) on the right-hand side of the third constraint and substituting it into the other equations to rewrite the linear program to

\[
\begin{align*}
z &= \frac{111}{4} + \frac{x_2}{6} - \frac{x_3}{8} - \frac{11x_6}{8} \\
x_1 &= \frac{3}{4} - \frac{x_2}{6} + \frac{x_3}{8} - \frac{11x_6}{8} \\
x_3 &= \frac{3}{4} - \frac{3x_2}{16} - \frac{x_5}{8} + \frac{x_6}{8} \\
x_4 &= \frac{63}{4} + \frac{3x_2}{16} + \frac{5x_3}{8} - \frac{x_5}{8} - \frac{x_6}{8}
\end{align*}
\]

There is still a nonbasic variable in the objective function with a positive coefficient. We increase this variable and pivot again to rewrite the linear program as:

\[
\begin{align*}
z &= 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3} \\
x_1 &= 8 + \frac{x_2}{6} + \frac{x_5}{6} - \frac{x_6}{3} \\
x_2 &= 4 - \frac{x_3}{3} - \frac{2x_5}{3} + \frac{x_6}{3} \\
x_4 &= 18 - \frac{x_2}{2} + \frac{x_5}{2}
\end{align*}
\]

There are no other variable we can change to increase the objective value. The basic solution to the linear program is \((8,4,0,18,0,0)\). The objective value from this solution is 28. We can now return to our original linear programing. The only variables in the original program are \(x_1\), \(x_2\), and \(x_3\). Using our basic solution these variables are \(x_1 = 8\), \(x_2 = 4\), and \(x_3 = 0\). Notice that these values give an objective value of 28 as expected. Note that our final solution is integral; although, this is not always the case.

8
Pivot ($N, B, A, b, c, v, l, e$)

// Compute coefficients for equation for the new basic variable $x_e$
// The variables with hats, like $\hat{b}_e$, will be the return values
$\hat{b}_e = b_l / a_{l,e}$
for each $j \in N - \{e\}$
do $\hat{a}_{e,j} = a_{l,j} / a_{l,e}$
end do
$\hat{a}_{e,l} = 1 / a_{l,e}$
// Compute coefficients for the other constraints
for each $i \in B - \{l\}$
do $\hat{b}_i = b_i - a_{i,e} \hat{b}_e$
for each $j \in N - \{e\}$
do $\hat{a}_{i,j} = a_{i,j} - a_{i,e} \hat{a}_{e,j}$
end do
$\hat{a}_{i,l} = -a_{i,e} \hat{a}_{e,l}$
// Compute the objective function
$\hat{v} = v + c_e \hat{b}_e$
for each $j \in N - \{e\}$
do $\hat{c}_j = c_j - c_e \hat{a}_{e,j}$
end do
$\hat{c}_l = -c_e \hat{a}_{e,l}$
// Compute new basic and nonbasic variable sets
$\hat{N} = (N - \{e\}) \cup \{l\}$
$\hat{B} = (B - \{l\}) \cup \{e\}$
return ($\hat{N}, \hat{B}, \hat{A}, \hat{b}, \hat{c}, \hat{v}$)

Figure 1: The Pivot algorithm. Adapted from [?].

3.2 Pivoting

Pivoting is a key, a perhaps the key, operation in the simplex algorithm. The pseudocode for the pivoting algorithm is given in Figure 1. The parameters passed to the algorithm are the matrices, vectors and integers in the concise representation of the slack form (the definition of the concise representation is on page 5).

3.3 The Simplex Algorithm, Finally!

The basic idea of the simplex algorithm is to choose a pair of entering and leaving variables. The pseudocode for the simplex algorithm is given in Figure 2. The entering variable, $e$ which is chosen on line 9, is a nonbasic variable (ie, its index is in $N$) with a positive coefficient in the objective function (ie, $c_e > 0$). The significance of the entering variable is that if we increase the value of the entering variable then we can increase the value of the objective function and that was our goal in the first place. Note that the algorithm on line 9 doesn’t give you any guidance on how to pick the entering variable. You just have to pick one, of possibly many, nonbasic variable in the objective function with a positive coefficient. You may decide to pick the nonbasic variable with the
Simplex \((A, b, c)\)

// convert the problem to slack form and check feasibility
// you may assume that the first basic solution is feasible
// this means you will only convert your problem to slack form
// this assumption is not true in general
\((N, B, A, b, c, v) = \text{InitializeSimplex}(A, B, c)\)

while there exists some \(j \in N\) such that \(c_j > 0\)
// \(e\) will be the entering variable

for each index \(i \in B\)
do if \(a_{i,e} > 0\)
then \(\delta_i = b_i/a_{i,e}\)
else \(\delta_i = \infty\)
// \(l\) will be the leaving variable
choose \(l \in B\) such that \(\delta_i\) is minimized
if \(\delta_l = \infty\)
then return “unbounded”
else \((N, B, A, b, c, v) = \text{Pivot}(N, B, A, b, c, v, l, e)\)
// set the nonbasic variables to 0 and everything else to the optimal solution
for \(i = 1\) to \(n\)
do if \(i \in B\)
then \(\bar{x}_i = b_i\)
else \(\bar{x}_i = 0\)
return \((\bar{x}_1, \bar{x}_2, \ldots \bar{x}_n)\)

Figure 2: The simplex algorithm. Adapted from [?].

biggest positive coefficient, or you might use some other method.

The for-loop in lines 10 through 14 determines how much we can change the value of the entering variable while allowing the basic variables (ie, variables with index in \(B\)) for each contraint to remain positive. In line 15, we pick the index of the basic variable with the tightest contraint (ie, let \(l = i\) such that \(\delta_i\) is the smallest of the \(\delta_s\)).

We repeat that process until either there are no more nonbasic variables with positive coefficients or we discover that the value of the objective function is unbounded (line 17). After completing the pivot operations, the only remaining task is to return the values of the nonzero basic variables.

The assumption in line 77 is made to simplify the project. Implementing the \text{InitializeSimplex} algorithm to ensure that the first basic solution is feasible is neither terribly difficult nor interesting so we will skip it. When you formulate the linear program for the project problem, check to see if the first basic solution is feasible. If it isn’t, then you need to think of a new formulation. The first solution to the “natural” formulation in slack form has a basic feasible solution. You can check to see if the first basic solution is feasible by setting all of the
nonbasic variables to 0 and determining if you still satisfy the constraints. For example, the constraint

\[ x_1 - x_2 \leq -3 \]

converted to slack form with \( x_1 \) and \( x_2 \) as nonbasic variables would have a basic solution with \( x_1 = 0 \) and \( x_2 = 0 \) which would violate this constraint because

\[ 0 - 0 \not\leq -3. \]