Chapter VI: Linear Programming I and Games

1 Introduction

Imagine that you are running some sort of enterprise, and have a number of possible activities you can undertake. Let us number these activities 1 through n. Suppose further, that these activities each involve use of various resources (numbered 1 through m) that you have available to you; a unit of activity j uses up am amount A_{ij} of resource i. Suppose you have an amount b_i of resource i available to you. Finally, suppose that these activities produce *value* for your enterprise, and in particular, a unit of activity j produces a contribution c_j to your total value.

Your task is to decide how much of each activity to perform, so as to maximize the value you produce, subject to the stated limitations on resources.

This problem, as stated, is a *linear program.* It is not surprising that this problem arises in many contexts and plays a fundamental role in the field of "Operations Research," which studies the means of making operations of all kinds better, according to whatever numerical criteria for "goodness" are appropriate.

If we let x_j be the number of units of activity j we want to produce, we have

$$x_j \geq 0, \text{ for each } j.$$

For each resource (and these can be each of the raw materials needed for manufacturing, as well as machine and staffing time for production), we have bounds

$$\sum_j A_{ij}x_j \leq b_i,$$

one bound for each i between 1 and m. (For example, we can add constraints of the form $x_j \geq m_j$ if we want to assure a minimum amount of activity j, or $x_j \leq M_j$ if we want to limit that activity.)

Under these constraints, all of whose left hand sides are linear functions of the variables x_j, we seek to maximize the value of these activities:

$$\textit{Maximize: } \sum_j c_j x_j.$$

This "objective function" is a linear function of the variables x_j. (It sometimes includes a constant term c_0, but the presence of c_0 clearly doesn't affect anything.)

If we let $\mathbf{A}$ denote the matrix of A_{ij}'s, $\mathbf{b}$ denote the column-vector of b_i's, $\mathbf{c}$ denote the row-vector of c_j's, and $\mathbf{x}$ denote the column-vector of x_j's, then the function to be

Figure 1: (a) A convex region; (b) a region which is not convex.

maximized can be written as $\mathbf{c} \cdot \mathbf{x}$ (the product of a row-vector and a column-vector); the constraints to be satisfied can be written as $\mathbf{x} \geq 0$ and $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$, where 0 here denotes the all-zeroes vector, and where we say that one vector is less-than-or-equal-to another if every entry of the first is less than or equal to the corresponding entry of the second (e.g., $(1, 3) \leq (2, 4)$).

We will now consider how to deal with such functions.

2 The Simplex Algorithm

2.1 Geometrical preliminaries

There are two ways to look at such problems, geometrically and algebraically. It is quite easy to visualize both the problem, and the *simplex algorithm* for solving it, geometrically. Fortunately, it is also easy to translate from the geometric picture to an explicit algebraic picture, which allows simple description and straightforward implementation of this simplex algorithm.

The simplest nontrivial problem occurs when there are two variables. Possible solution points then correspond to points in the plane. A linear constraint corresponds to a line in the plane, and the restriction that the solution must lie on one side of that line. The function to be maximized, called the *objective function,* corresponds to a direction in the plane. The problem then consists of finding a point in the region defined by the constraints, having the largest component in the direction defined by the objective function. This will be the point, which, among those obeying the constraints, has the largest dot product with a fixed vector pointing in the direction of the objective function.

A region is *convex* if whenever we draw a line between two points in the region, the line we draw also lies entirely inside the region. It is easy to see that the region defined by linear constraints is convex: if two points obey any constraint, so do any points on the line between them.

The region within and allowed by the constraints is called the *feasible region* for the given problem. It may be bounded, in which case it is a convex polygon (in two dimensions) or may be unbounded. The constraints may be such that there is no feasible region, or so that region is unbounded and the objective function can grow arbitrarily large within the feasible region. In either of these cases the maximization program is said to be *infeasible.*

It is easy to see that with two variables, if the program is feasible, there is always a maximizing solution point; this point will always lie at a vertex of the feasible region. This is also true with three variables, where the feasible region is some subset of three-dimensional space: each constraint restricts the feasible region to lie on one side of a plane. The feasible region is therefore like some sort of a cut diamond, perhaps of infinite extent; if it is bounded, it is a convex polyhedron. Again, we seek a point in the feasible region that has the largest possible component in some direction. As before, we can always find such a point at one of the vertices of the feasible region.

It is somewhat less easy to visualize problems in more than three dimensions, but the intuitions we develop in low-dimensional cases apply quite well. The constraints restrict the solution to lie on one side of what is called a *hyperplane,* which is to the higher-dimensional space what a line is to two-dimensional space or a plane is to three-dimensional space. Again we seek a point in the feasible region (which, if bounded, is called a *convex polytope*) that has the largest component in some direction.

A *vertex* v of a polytope P in n dimensions is the intersection point of n of its bounding hyperplanes, that lies in the boundary of P. An *edge* of P is that portion of the intersection of $n-1$ of the bounding hyperplanes that lies between two of its vertices. In two or three dimensions these are the ordinary vertices and edges of the polygon or "cut diamond" that form the polytope. (Actually, these definitions are not quite correct; we will fix them shortly.)

The points of a convex polytope P are called *convex combinations* of its vertices. That is, every point q in P can be represented as a linear combination of the vertex points of P, with positive coefficients that sum to one. Therefore, any point z in a convex polytope is an *average* of vertex points, where the vertex points are given weights in this average by their coefficients in this representation of z. The value of the objective function F at z is, by linearity of the objective function, the average of the values of F on the vertex points, with the same weighting on them. But an average of quantities is never bigger than the largest of the quantities averaged. All of this tells us that the largest value of the objective function must occur on a vertex, if the feasible region is a bounded polytope. The same conclusion actually holds whenever the program is feasible, even when the region is not bounded (the proof of which is left to you). If there is a feasible point on which the objective function is greater than its value on any vertex, then the objective function is unbounded (the proof of this is also left to you).

Now we can explain what is wrong with the definition of "vertex" given above. Suppose we have a polytope consisting of those points (x, y, z) that satisfy the inequalities $x \geq 0$, $y \geq 0$, $z \geq 0$, $x \leq 1$, $y \leq 1$, $z \leq 1$ together with the (redundant) constraint $x + y \geq 0$. That is, the polytope is a unit cube. Every point on the edge of the cube joining $(0,0,0)$ to $(0,0,1)$ lies on all three of the constraint-planes $x = 0$, $y = 0$, $x + y = 0$. But should we on this account call points like $(0,0,\frac{1}{2})$ vertices of the cube, just because they lie on three bounding planes? Surely this is not what we mean by vertices! We could try to patch the original definition by requiring that the n planes on which v lies should form a non-degenerate arrangement. However, a better approach is to say that a point is a vertex (or *extreme point*) of a region if it belongs to the region but cannot be written as an average of two other points in the region. Thus, $(0,0,\frac{1}{2})$ is not a vertex of the cube, because it is the average of $(0,0,0)$ and $(0,0,1)$. It is a basic theorem of convex geometry that any point belonging to a bounded convex region is either an extreme point of the region or else can be written as a (weighted) average of extreme points.

In the same way, the definition of "edge" given above doesn't work so well when the bounding planes have degeneracies. To be fully general, we should opt for a different definition entirely, and say that a point in a polytope P is part of an edge if there is at most a 1-parameter family of ways to write it as the average of two points in P.

Not every set of n bounding hyperplanes determine a vertex, but every vertex is the (unique) intersection point of some set of n bounding hyperplanes. Thus, if there are m constraints in our linear program, so that (allowing for the extra constraints that all n variables must be non-negative) there are $m + n$ bounding hyperplanes, there are exactly $\binom{m+n}{n}$ ways to choose n of the hyperplanes, and hence there are at most this many vertices of the feasible region. If this region is known to be finite, then one brute-force procedure for solving the problem is to examine the value of the objective function at each of these vertices, and find the vertex for which it is greatest. But if the feasible region is permitted to be infinite, then even this approach does not suffice: for, the objective function might or might not be bounded, and merely examining all the vertices won't tell us which is the case. We need another idea.

Exercise 6.1 Write a proof of the two unproved assertions made in subsection 6.1.2, in the paragraph that introduces the notion of convex combinations of points.

2.2 The simplex algorithm

Imagine a polyhedron lying at the bottom of an empty tank, and imagine a bug sitting on a vertex of the polyhedron. Assume that the bug can move only along edges of the polyhedron. If we slowly fill the tank with water, the bug will attempt to avoid drowning by seeking ever-higher positions on the polyhedron. Suppose we stop filling the tank at the precise moment when only one point of the polyhedron is exposed. Will the bug make it to that point? Or might he have drowned, having gotten cornered-in someplace by the rising tide? (Remember, the bug is not allowed to take a downward switch-back.)

Of course, the answer to this question depends on both the shape of the polyhedron and the bug's strategy. But if the polyhedron is *convex*, then any strategy will do: as long as the bug keeps seeking a higher foot-hold, it doesn't matter which edge he takes when faced with a choice—he will end up at the apex of the polyhedron. We will not prove this fact, here, but if you try experiments (mental or real), holding cubes and other convex polyhedra at various angles and picturing the path of the bug, you will come to believe it.

More generally, if F is some linear function of the three coordinates of a point in 3-space, the bug can find the vertex of the polyhedron that maximizes F by starting at any vertex and following edges along which the value of F increases until he gets stuck (i.e., until he finds no edge emanating from his current vertex along which F increases). To see this, imagine tilting the picture until the planes $\{(x, y, z) : F(x, y, z) = \text{constant}\}$ are horizontal.

What worked above for polyhedra works just as well for polytopes. It is an example of a *greedy algorithm*: a procedure that attempts to optimize some quantity by making short-sighted improvements of a local nature. For most problems, greedy algorithms do not find optimal solutions. (Imagine a bug on a non-convex polyhedron, stranded on a path that takes it to the shorter of two peaks, after the water level has risen high enough to turn the peaks into two separate islands!) However, for linear programming, a greedy approach *does* work; this is at the heart of the simplex method.

We now give a geometric description of the simplex algorithm. We start at a vertex of the polytope. We then examine the edges that touch this vertex, choose one along which the objective function increases, and move to the vertex at the other end of that edge, changing coordinates to make it the origin. We repeat this procedure again from the new origin to a vertex at which the objective function is even bigger, until there are no edges along which the objective function increases, in which case we have a solution; or until we find an infinite edge along which the objective function increases to infinity, in which case the program in infeasible. This is the *simplex algorithm.*

At this point we make the following remarks:

1. We have not yet completely described the procedure, because we have not specified which edge to choose, in leaving a vertex, if the objective function increases along several different edges. There are in fact many variations in the simplex algorithm that use different rules for deciding which way to go when there is a choice: here are several:

(a) go at random; pick among available choices arbitrarily.

(b) go along the first edge you find along which the objective function increases.

(c) go in the direction whose dot product with the objective function is largest.

(d) go along the edge containing the current origin whose other endpoint makes the objective function as large as possible.

(e) choose some secondary direction and choose the edge whose other vertex has the largest dot product in this direction.

(f) start with the vertex of the feasible region that *minimizes* the objective function, or in other words maximizes its negative; then start "rotating" this modified objective function (around some axis) until it coincides with the original objective function. If for each intermediate stage of the rotation you keep track of which vertex of the feasible region maximizes the rotating objective function, when the rotation is done one will have the desired solution. (Believe it or not, this is a common method, and it is easy to do; this is the method we will analyze in detail.)

All of these are formulations of the *simplex algorithm*.

2. As a general procedure what we have said so far has a shortcoming; what if there is no feasible region at all? what would have happened? how could we have gotten anywhere or started this procedure? In any case, how do we start if the origin of our original coordinate system is not a vertex of the feasible region?

3. What we have said so far is rather vague. It is all very well to speak of marching from vertex to vertex on a polytope, but we are not elves capable of walking on polytopes; we must define what it means to *be at* a vertex of the feasible region, and *go to* another. The obvious way is to keep track of the coordinates of our moving point, relative to a fixed origin. A less obvious way is to keep changing our coordinate system, so that the moving point is always at the origin. This is the approach we will adopt. It requires that we modify the matrix $\mathbf{A}$ and the vectors $\mathbf{b}$ and $\mathbf{c}$ many times, to accord with our repeated change-of-variables, but it has a certain conceptual simplicity. Moreover, once we have understood the approach, we can go back and recast the algorithm in a purely numerical way that we can make sense of without actually thinking of moving the origin around.

4. The act of moving from vertex to vertex, which is called *pivoting*, will as we shall see, take on the order of nm steps (of multiplications and additions) if

there are n positive variables and m additional constraints. The efficiency of the simplex algorithm is a function of the difficulty of this pivoting task, of the difficulty of the task of deciding which edge to use for each pivot, and of the number of pivots required to solve the problem. Just how many pivots are required for problems of this kind depends on how we go about pivoting, and is hard to know in general. There is much folklore on this subject which we will discuss later.

5. In the most general problem of the form we are considering, some of the constraints may be equalities and some inequalities; some of the variables may be constrained to be positive; and some may not be constrained at all. Having all of these different possibilities tends to complicate notation, but actually does not make the problem more difficult in theory. We will therefore begin by assuming that all constraints are inequalities, and all variables are constrained to be positive or zero. We can do this without serious loss of generality, since there is no problem in generalizing what we do to include equality constraints and unconstrained variables.

We will now respond to the issues raised in the foregoing remarks, by writing the problem in algebraic form, and describing the simplex algorithm explicitly algebraically, in each of the variations mentioned.

We will handle the problem of the second remark by revising the original problem so as to make the origin of coordinates a vertex of the polytope of the revised problem. We will do this so that when we try to solve the revised problem, we obtain either a feasible solution to the original problem or a proof that the old feasible region was empty (i.e., no feasible solution exists).

A key idea underlying the simplex method is that every vertex of the feasible region is an intersection of n constraint hyperplanes, so we can take these as the coordinate-planes of an oblique coordinate system, and thereby give the current vertex coordinates $(0, 0, \ldots, 0)$ in the new coordinate system. When we move along an edge of the polytope to a new vertex, we retain $n - 1$ of the bounding hyperplanes and trade in the nth for a new one. (If you imagine that the nth bounding hyperplane, instead of being simply replaced by a different bounding hyperplane, rotates around the edge joining the two vertices until it comes into the correct position, then you can see how the term "pivot" arose.)

We will then return to discuss the question: how may pivots will occur as we move towards the optimal vertex (or vertices)?

Figure 2: The feasible region of a linear program.

3 The Algebraic Formulation

3.1 Slack variables

We consider a problem with n variables, $x_1, \ldots, x_n$ and m constraints. We denote the coefficient of x_j in the ith constraint as A_{ij} and the constant term in that constraint as b_j. The coefficient of x_j in the objective function we denote by c_j.

Our problem therefore is described as follows:

$$\begin{gathered}\text{Maximize } \sum_j c_j x_j \\ \text{subject to } x_j \geq 0 \text{ for } 1 \leq j \leq n \\ \text{and } \sum_j A_{ij} x_j \leq b_i \text{ for } 1 \leq i \leq m.\end{gathered}$$

Here is a simple example. Maximize $x_1 + 2x_2$, subject to $x_1 + x_2 \leq 1$, $x_2 - x_1 \leq 0$, $x_1 \geq 0$, $x_2 \geq 0$. Then we can describe A by the matrix:

$$\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix},$$

with $\mathbf{b}$ given by $(1, 0)$ and $\mathbf{c}$ by $(1, 2)$. The feasible region is the shaded region in Figure 2.

In this form, the problem involves m inequalities in n variables (leaving aside momentarily the positivity constraints on the n variables, which are special). We rewrite our m inequalities as m equations in $n+m$ variables. We do this by introducing a *slack variable* s_i for each constraint, and rewriting the constraints as:

$$\sum_j A_{ij} x_j + s_i = b_i \text{ for } i \leq m$$

so that the old inequality constraints now read:

$$s_i \geq 0, \text{ for } 1 \leq i \leq m.$$

Now we have $n+m$ variables, x_j's and s_i's, each of which must be positive or zero, and there are m equality constraints that these must obey.

Notice that in matrix form, the equations there have an identity matrix part in the s-variables (the s_i's) and the original $\mathbf{A}$ matrix terms in the x-variables (the x_j's):

$$\begin{pmatrix} \mathbf{I} & \mathbf{A} \end{pmatrix} \begin{pmatrix} \mathbf{s} \\ \mathbf{x} \end{pmatrix} = \mathbf{b}. \tag{1}$$

The matrix $(\mathbf{IA})$ is reminiscent of the $[I, Z]$ matrices we saw when discussing matrix codes.

3.2 Row-reduction

(1) is what is called an underdetermined system of linear equations: there are $n+m$ variables but only m constraints. When we have such a system of linear equations, we can manipulate them (add a multiple of one to another, etc.) without changing their content. In particular we can pick out any m of the variables and *row-reduce* the equations with respect to those variables. That is, we can rearrange the equations so that each of the m picked variables is "solved for" in one equation; so that it appears in only that equation and does so with coefficient one. The equations as given in (1) are row-reduced for the set of s-variables when we begin.

Our m equations may have many forms, as we have just noted. We connect the algebraic forms of the equations now under discussion, with our geometric picture, by *associating to each vertex v of the polytope, any form of these equations that is row-reduced with respect to all variables that are non-zero at v.* (Remember that a vertex is a feasible point that lies on n constraint hyperplanes: in our present notation this means that a vertex is a point at which (at least) n of the x's and s's vanish, and the others, of which there are at most m, are positive.) Thus, the origin $x_1 = x_2 = \ldots = x_n = 0$ is associated with the original form of the equations, given by (1).

Suppose that we have some row-reduced form of the equations with matrix $(\mathbf{I}\,\mathbf{A}')$, and we set the variables corresponding to the $\mathbf{A}'$ part of the $(\mathbf{I}\,\mathbf{A}')$ matrix to zero, we are able to determine the values of the remaining variables; the part of the matrix involving these variables is just the identity matrix. Let us label the former variables $x_1', x_2', \ldots$ and the latter variables $s_1', s_2', \ldots$. Then we have

$$\begin{pmatrix} \mathbf{I}' & \mathbf{A}' \end{pmatrix} \begin{pmatrix} \mathbf{s}' \\ \mathbf{x}' \end{pmatrix} = \mathbf{b}' \tag{2}$$

so that

$$\mathbf{I}'\mathbf{s}' + \mathbf{A}'\mathbf{x}' = \mathbf{b}'.$$

If we set $\mathbf{x}' = 0$ (defining our origin), we can read off the values of the remaining variables in $\mathbf{s}'$:

$$\mathbf{s}' = \mathbf{b}'.$$

A row-reduced $(\mathbf{I}\,\mathbf{A}')$ form tells us the values of all variables when those variables that correspond to the $\mathbf{A}'$-part are set equal to zero; that is, the row-reduced form tells us the values of all the variables at the *origin* in the coordinates corresponding to the $\mathbf{A}'$ part of the matrix.

Not every row-reduced $(\mathbf{I}\,\mathbf{A}')$ form of the matrix determines the origin to be a vertex of the polytope. (That is, not every set of n bounding hyperplanes intersect in a vertex of the feasible region.) It is necessary in order for the origin to be a vertex that it itself be a feasible point; i.e., that all the constraints are obeyed at it. This will be so when all the components of $\mathbf{b}'$ are positive. At the beginning of the simplex algorithm, $\mathbf{b}'$ is just the original constraint-vector $\mathbf{b}$, and so if all entries of $\mathbf{b}$ are positive (as is usually the case), then the origin starts out being a feasible point. In this case, every vertex we subsequently take as the origin will also be a feasible point (by virtue of the nature of the pivoting-operation).

Once we have a row-reduced $(\mathbf{I}\,\mathbf{A}')$-form of the linear program and an associated vertex to serve as the origin, we can use our m equations to rewrite the objective function in terms of the n variables corresponding to $\mathbf{A}'$. Since these variables all vanish at our vertex, the value of the objective function there will be its constant term, c_0, when written in these terms.

If a vertex of a feasible region has more than n variables vanishing, it is said to be a *degenerate vertex* of the problem, and the problem is said to be *degenerate*. This occurs when an extra hyperplane, or several such beyond the usual n of them, pass through a vertex of the feasible region. We will assume here that our problem is not degenerate. It is easy to determine when we are in a degenerate situation, and if we are, there are standard steps that can be taken to remove the degeneracy: one perturbs the problem slightly so that each extra hyperplane is displaced a little from the vertex, whenever this occurs.

If the problem has no degeneracy, then at each vertex of the feasible region exactly m variables fail to vanish and we can associate the vertex with a unique form of the equations: the one that is row-reduced with respect to those variables. The statement that the origin in the x's is feasible says that no b_i is strictly negative; that the origin is feasible and non-degenerate is the statement that all the b_i's are strictly positive. Our original form of the equations is already row-reduced, with the s_i's the row-reduced variables, and the x_j's the variables that vanish at the vertex (origin) corresponding to this form. Thus the origin in the x-variables, if it is a vertex of the feasible region, is the point associated with the starting form of the constraint equations.

3.3 Pivoting

Let us assume for the moment that the origin is a vertex of the feasible region. If it is not, we will (by a process to be described later) define a new problem for which the origin is feasible, so we will get to this situation in any case.

The simplex algorithm then starts from a vertex v (taken to be the origin). We then test the edges emanating from v, to find one along which the objective function increases. Algebraically, this translates into the statement that we examine the various variables that vanish at our vertex, and choose one, say x_k, whose coefficient, c_k, in the objective function written in terms of the x-variables is positive. The edges along which this variable is allowed to become positive (while the other variables that vanished are held constant) contributes a positive amount to the objective function when c_k is positive (since it contributes $c_k x_k$ to the objective function).

We can let this variable, x_k, increase until we hit one of the constraints whose s-variable did not vanish at v. This will occur for a constraint q for which $A_{qk} > 0$ and among these, one for which b_q/A_{qk} is the smallest, since the amount of slack associated with the qth constraint changes by $A_{qk}x_k$ as x_k increases, and when x_k reaches b_q/A_{qk} with the other x's still zero, s_q will vanish. If there are no positive A_{qk}, then one can increase x_k indefinitely, raising the objective function to infinity, without violating any constraint; in that case the objective function is unbounded in the feasible region.

Thus by choosing an x_k having a positive c_k and finding the row with the smallest positive ratio of b_q/A_{qk} in its column, we can determine an edge to move along, and where the vertex at its other end is. One performs a pivot, by changing the form of the m equations so they are row-reduced for all the x's but x_k, and for s_q in place of x_k. This is done by solving the qth equation for x_k in terms of the other x's and s_q: substituting the result for x_k wherever else it appears in M, and in the objective function, and treating s_q from now on as if it were an x (we will rewrite the equations treating s_q as the new x_k variable, and vice versa).

One can write explicit formulae for the changes in the constraint matrix, the b_i's, and the c_j's caused by a pivot. They are as follows, where the kth x-variable after the pivot is identified with the old qth s-variable, and vice versa:

$$A_{qk} \text{ becomes } 1/A_{qk}$$

(we divide the whole qth row by A_q and then make the s_q variable which had coefficient 1 before the pivot, and $1/A_{qk}$ after it, into the new x_k variable),

$$A_{qj} \text{ becomes } A_{qj}/A_{qk} \text{ for } j \text{ other than } k$$

(from solving the q equation for x_k),

$$A_{jk} \text{ becomes } -A_{jk}/A_{qk} \text{ for } j \text{ smaller than } q$$

(from using the qth equation to eliminate x from the jth equation and then switching the identity of s_q and x_k),

$$A_{ij} \text{ becomes } A_{ij} - A_{ik}A_{qj}/A_{qk}$$

(from substituting for x_k wherever else it occurs in the equations),

$$b_q \text{ becomes } b_q/A_{qk}$$

(from solving for x_k),

$$b_i \text{ becomes } b_i - b_q A_{ik}/A_{qk}$$

(from eliminating),

$$c_k \text{ becomes } -c_k/A_{qk}$$

(from switching and eliminating),

$$c_j \text{ becomes } c_j - c_k A_{qj}/A_{qk}$$

(from eliminating x_k from the objective function), and

$$c_0 \text{ becomes } c_0 + b_q c_k/A_{qk}.$$

(Here c_0 is the constant term in the objective function.)

Our first example is degenerate at the origin, since $b_2 = 0$. If we pivot on x, the appropriate equation to use in the pivot is the first. (Since A_{21} is negative we would not pivot about 21 so that the degeneracy of the second equation is no bother, if we instead tried to pivot with the variable y, we could go only a distance zero, because of the degeneracy; it leads to the risk of pivoting around in circles; if we are ever forced to pivot using a degenerate row (having $b_i = 0$) we instead take evasive action.)

Original form:

original variables current variables	s_1	s_2	x x_1	y x_2	b
row 1	1	0	1	1	1
row 2	0	1	-1	1	0
c			1	2	0

after (1 1) pivot:

original variables current variables	s_1 x_1	s_2	x	y x_2	b
row 1	1	0	1	1	1
row 2	1	1	0	2	1
c	-1			1	1

after (2 2) pivot:

original variables current variables	s_1 x_1	s_2 x_2	x	y	b
row 1	$1/2$	$-1/2$	1	0	$1/2$
row 2	$1/2$	$1/2$	0	1	$1/2$
c	$-3/2$	$-1/2$			$3/2$

The answer then is that the maximum (here d_0) is $3/2$, which occurs for $x = 1/2$ and $y = 1/2$.

Degeneracy occurs when one of the b_i's becomes zero. We can get around that by letting any such b_i be .001 (or some other small number) whenever it is zero. When we pivot we will then go a small finite distance and the objective function will change slightly, but now we can never go back and therefore cannot go in circles. When the final answer is found we can revert to our original vector $\mathbf{b}$, and reverse the effects of our small change of the problem.

Exercise 6.2 Write a program (or pseudocode) that performs the simplex algorithm when the origin is a feasible point.

4 Feasibility

The simplex method we have been discussing starts with the form of our linear equations that corresponds to the origin of our x-variables. This is a good place to start if this origin is a vertex of the feasible region, but it is not a possible place to start if the origin is not a vertex. In general, the origin will not satisfy all constraints; though n

constraints are equality there, there may well be other constraints that are violated at the origin, and it will then not be a vertex. In algebraic terms, this happens when some of the b_i's are negative: thus the equations obtained by setting all the x_j's to zero will not be consistent with the positivity constraints on the slack variables.

What we do in this case is to introduce a new variable, x_{n+1}, in such a way that:

1. the origin in all $n+1$ variables is definitely a solution to the new problem;

2. when $x_{n+1} = 1$ the problem reverts to the original one; and

3. there is a constraint $x_{n+1} \leq 1$.

The origin in this case is then not only a feasible vertex but the one that minimizes the variable x_{n+1}, which we will treat as our *first* objective function; what we then do, following plan f of our first remark, is to rotate this objective function by 180 degrees, around from $-x_{n+1}$ through our actual objective function to x_{n+1} always staying at the maximum of the rotating objective function. If we ever reach the hyperplane $x_{n+1} = 1$ we have our solution; if we cannot reach it, the original program was not feasible.

Let us see now that we can introduce this new variable so as to make the region feasible (all b_i's positive) without changing the problem on the hyperplane on which the new variable is 1.

What we do is to take any and every negative b_i and change it to 1, by adding a term $(1 - x_{n+1})(b_i - 1)$ to the left hand side of its constraint equation. Notice that this has no effect at all when x_{n+1} is one, yet changes the inhomogeneous b_i term to 1.

Consider the following example, with three variables

$$
\begin{aligned}
x_1 - 3x_2 + x_3 &\leq -2 \\
-2x_1 - x_2 - x_3 &\leq -3 \\
x_1 + x_2 + x_3 &\leq 1
\end{aligned}
$$

maximize $x_1 + x_2$.

We introduce the variable x_4 and have as constraints:

$$
\begin{aligned}
x_1 - 3x_2 + x_3 + 3x_4 &\leq 1 \\
-2x_1 - x_2 - x_3 + 4x_4 &\leq 1 \\
x_1 + x_2 + x_3 &\leq 1 \\
x_4 &\leq 1
\end{aligned}
$$

We know that the origin in all four variables maximizes the "objective function" $-x_4$ in the feasible region, and will change the objective function by rotation according to:

$$F(a) = -x_4 \cos(a) + (x_1 + x_2) \sin(a)$$

always staying at a maximum of this function, until either x_4 becomes 1, or a becomes 180 degrees; at which point we have either a solution or a proof that there is none. How do we do this? Geometrically, we can think of the problem as follows:

We are at a vertex v, which is a maximum point for our objective function, F. We rotate that function; when should we leave v and go along an edge e to some other vertex v' , given that we want to stay at the maximum of F? the objective function F defines a vector $c(a)$. When $(v - v') \cdot c'(a)$ is positive v will have a larger value of the objective function than v': if this dot product is negative, we will have a larger value of F than v. *Thus the change in solution must occur when $v - v'$ has zero dot product with (is perpendicular to)* $c(a)$. Therefore, we can determine which edge to use to leave v, by finding which is the edge that first becomes perpendicular to $c(a)$ as we rotate F.

In algebraic terms, we need only find which of the current set of x-type variables first switches from a negative to positive coefficient, as we increase the angle a.

Let us perform this procedure explicitly for our example; initially after introducing slack variables $s_1, \ldots, s_4$ we can write our linear program as:

s_1	s_2	s_3	s_4	x_1	x_2	x_3	x_4	b
1	0	0	0	1	-3	1	3	1
0	1	0	0	-2	-1	-1	4	1
0	0	1	0	1	1	1	0	1
0	0	0	1	0	0	0	1	1
				s	s	0	$-c$	

We may now pivot on either x_1 or x_2; we choose the latter which causes us to pivot on the third row, obtaining the new "tableau":

s_1	s_2	s_3	s_4	x_1	x_2	x_3	x_4	b
1	0	3	0	4	0	4	3	4
0	1	1	0	-1	0	0	4	2
0	0	1	0	1	1	1	0	1
0	0	0	1	0	0	0	1	1
		$-s$		0		$-s$	$-c$	s

Here s is short for $\sin(a)$ and c_j for $\cos(a)$. As we continue to rotate, when a becomes 90 degrees, the coefficient of x_4 vanishes, and we will therefore pivot on the corresponding column of the tableau. This will cause a pivot on the second row, and writing $-\cos(a)$, which is now positive, as c_j, we get:

s_1	s_2	s_3	s_4	x_1	x_2	x_3	x_4	b
1	$-.75$	2.25	0	4.75	0	2	0	2.5
0	.25	.25	0	$-.25$	0	0	1	.5
0	0	1	0	1	1	1	0	1
0	$-.25$	$-.25$	1	.25	0	0	0	.5
	$-.25d$	$-s-.25d$		$.25d$			$s+.5d$	

We must again pivot, this time on the x_1 column, because its coefficient, $d/4$, has now become positive: we get

s_1	s_2	s_3	s_4	x_1	x_2	x_3	x_4	b
4/19	$-3/19$	9/19	0	1	0	8/19	0	10/19
1/19	4/19	7/19	0	0	0	2/19	1	12/19
$-4/19$	3/19	10/19	0	0	1	11/19	0	9/19
$-1/19$	$-4/19$	$-7/19$	1	0	0	$-2/19$	0	7/19
$-d/19$	$-4d/19$	$-s-5d/38$				$-s-8d/19$		...

At this point we are done; all of the coefficients in the objective function are now negative, and stay negative as we rotate a to 180 degrees. The solution corresponding to this form of the equation is $x_1 = 7/19$, $x_2 = 9/19$, $x_3 = 0$, $x_4 = 12/19$. Since x_k is not 1, the original program was not feasible. (We could have observed this by noticing that twice the third equation plus the second produces a contradiction among the positivity constraints.)

In this method, the objective function coefficients will have both s and c or d terms in them, and we pivot at the values of a at which these first become positive. Notice that in our example both x_1 and x_2 had their coefficients in the objective function become positive at the same time, at $a = 0+$ at the first stage of the algorithm. There is still a choice as to which pivot to make at vertices at which this happens. It is consistent with this method to choose an arbitrary preference ranking among the x_k's when this happens, as to which variable to use to pivot, and to give preference among the s_i's and x_k's again fairly arbitrarily, if this phenomenon should happen at a second vertex.

Exercise 6.3 Solve the following linear program using the method described above (write a program or solve it by hand).

1	0	0	0	1	-3	1	3	-1
0	1	0	0	-2	-1	-1	4	-1
0	0	1	0	1	1	1	0	1
0	0	0	1	0	0	0	1	1
				1	1	0	1	

Exercise 6.4 Express each of the pivoting rules for alternatives c through e in explicit algebraic terms. That is, state the algebraic condition that determines which among several possible pivots is to be performed in the simplex algorithm.

Some Remarks: In practice, it often happens that there are some variables that are not constrained to be positive; and there are constraints that are equalities rather than inequalities. Both of these represent slight simplifications of the problem; as we now note.

If one of our constraints is an equation, this means geometrically, that all points are confined by it to being inside than on one side of a hyperplane of the original space. It in effect reduces the dimension of the problem by one. It does so algebraically most of all; since the constraint is an equation, we need introduce no slack variable to make it one. We can therefore pick out any of the original variables that appear in it and make it an honorary slack variable for the equation; by solving it for that variable, and using it to eliminate the variable from the other constraints.

If one of the variables is unrestricted in range, so that it need not be positive, then its being zero is no significance at all in our problem. We can solve any of our equations for this variable, substitute for it in the others, and then disregard the equation solved in determining future pivots: since we do not care whether the b_i for that equation is positive or negative. When this happens in effect the number of constraints that we need worry about is reduced by one.

There is a possible disadvantage to choosing an arbitrary variable as slack for an equality constraint or to choosing an arbitrary equation to solve for an unrestricted variable, and that is, either of these may cause b_i's to become negative, so that the origin is no longer a feasible vertex after the dimension of the problem is reduced by these actions. Of course this is no problem if some b_i's were negative to begin with. Since our technique for handling negative b_i's requires the addition of one constraint and one variable, the indicated procedure will help if undertaken before introducing the new variable and constraint, and if there are several unrestricted variables or equality constraints.

An alternative procedure is to choose among the equations to solve for an unrestricted variable, or variables to solve for in an equation constraint to find the one that does not make b_i's negative. For a variable, this will be the constraint with the smallest ratio of b_i to the magnitude of the coefficient of the variable. For a constraint, one can look for a variable in it to make into its slack that will leave all relevant b_i's positive, but one might fail to find one.

5 Duality

Suppose we have a linear program in the standard form, which is:

$$\text{Maximize } \sum_j c_j x_j$$

$$\text{subject to } x_j \geq 0 \text{ for all } j \leq n$$

$$\text{and } \sum_j A_{ij} x_j \leq b_i \text{ for } i \leq m.$$

We have seen, that we can solve such a system of constraints by the simplex algorithm, which involves performing a sequence of *pivot operations*, each describable as follows: for some q and k:

$$A_{qk} \text{ goes to } 1/A_{qk},$$

$$A_{qj} \text{ goes to } A_{qj}/A_{qk} \text{ for } j \text{ not } k,$$

$$A_{ik} \text{ goes to } -A_{ik}/A_{qk} \text{ for } i \text{ not } q,$$

$$A_{ij} \text{ goes to } A_{ij} - A_{ik}A_{qj}/A_{qk} \text{ for } j \text{ not } k \text{ and } i \text{ not } q,$$

$$b_q \text{ goes to } b_q/A_{qk},$$

$$b_i \text{ goes to } b_i - b_q A_{ik}/A_{qk},$$

$$c_k \text{ goes to } -c_k/A_{qk},$$

$$c_j \text{ goes to } c_j - c_k A_{qj}/A_{qk}, \text{ and}$$

$$c_0 \text{ goes to } c_0 + b_q c_k/A_{qk}.$$

We have a solution to this linear program when all the b_i's are positive and all the c_j's are negative. There is a second linear program that can be described by the same $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$, that is transformed in the same way by pivot operations, and which has a solution when the b_i's are positive and c_j's are negative, which solution will have the same value of c_0. In consequence, the extremal value of the objective function at the solution will be the same for this dual problem as it is for the first.

The *dual* is the problem we might write down if we had a terrible memory or couldn't tell right from left and tried to write down the *primal* problem from memory. It has m variables, which we will write as y_i's, and n constraints; it has the roles of b_i's and c_j's reversed and has the inequalities in the constraints reversed, and we minimize the objective function rather than maximize it.

Here it is explicitly:

$$\text{Minimize } \sum_i b_i y_i$$

$$\text{subject to } y_i \geq 0 \text{ for } 1 \leq i \leq m$$
$$\text{and } \sum_i y_i A_{ij} \geq c_j \text{ for } 1 \leq j \leq n.$$

This is not written in the standard form used for the *primal* problem. If you prefer that form, here is what the dual looks like:

$$\text{Maximize } \sum -b_j y_j$$

$$\text{subject to } y_j \geq 0 \text{ for all } j \leq m,$$
$$\text{and } \sum -A_{ji} x_j \leq -c_i \text{ for } i \leq n.$$

In this form it is the $-b_i$'s that play the role that the c_j's play in the primal, and $-c_j$'s play in the primal, and $-c_j$'s play b_i's role. Thus the conditions that the b_i's be positive and the c_j's negative is the solution for the primal and dual both.

It is easy to see that the pivot operations have the same effect on the primal and dual tableaux, since the operation of taking the dual consists of changing all signs and transposing the A matrix. If we look at what the pivot actually does as listed above, we see that the terms that change sign upon transposition are precisely those that change sign of everything (in particular A_{qj}, A_{jk}, b_q, and c_k). The constant term in the objective function in the standard form of the dual also has a changed sign, being an odd function of the A_{ij}'s, b_i's, and c_j's; therefore the solution maximum for $-\mathbf{b}\cdot\mathbf{y}$ is the negative of the primal maximum; the minimum value of $\mathbf{b}\cdot\mathbf{y}$ in the dual is exactly the maximum value of $\mathbf{c}\cdot\mathbf{y}$ in the primal, as we have claimed.

In fact verifying that pivots do the same thing to the primal and dual and that the condition for a solution is the same for the two problems is a proof of the following *duality theorem*:

Theorem 1 *(Duality) The primal and dual programs are each feasible if and only if the other is, and the objective functions take the same value at the optimal solutions for both.*

Proof. Suppose one problem has a solution; pivot from the original form until you get to it; the pivots can be applied identically to the dual problem, and one again has the b_i's positive and c_j's negative for it, hence a solution for it as well. As we have noted the objective function $\mathbf{b}\cdot\mathbf{y}$ has been transformed in exactly the same way as $\mathbf{c}\cdot\mathbf{x}$ in reaching this point, and these objective functions are therefore the same there.

We have already associated a variable, namely a slack variable s, with each constraint. Upon introducing the dual variable y associated with the same index, we have a pair of variables for each constraint. Similarly, upon introducing a *dual slack variable*, t_j for each slack constraint (so that the constraint requires that variable to be positive) we have an x and a t for each of our original variables.

Pivoting to find a solution vertex, as we have seen, involves selecting m of the x's and s's to row-reduce our equations with. Pivoting in the dual amounts to the same thing with the dual variables, the y and the t's.

We now ask, what is the interrelation between the dual solution and the primal one at the solution point? Initially the row-reduced primal variable, the s's are precisely those that pair with y's which are the dual variables that are originally not row-reduced, and which vanish at the dual origin. Making the identical pivots in the the primal and dual preserves this situation: the variables that don't vanish in the primal solution (these are those variables for which the primal equations are row-reduced at the solution, since we are assuming that there are no degeneracies) are precisely those that correspond to dual variables that do vanish at the dual solution. It is therefore possible to read off the dual solution from the primal one (or vice versa) if there is no degeneracy: set those variables in the dual equal to zero that are dual to non-vanishing variables at the primary solution.

Another sometimes useful property of the dual is that the value of the objective function $\mathbf{b} \cdot \mathbf{y}$ at any feasible dual point is an upper bound to the value of the primal objective function at any of its feasible points.

We can see this by evaluating $\mathbf{x} \cdot \mathbf{A} \cdot \mathbf{y}$ in two ways.

If we write the primal and dual equations as

$$\mathbf{A} \cdot \mathbf{x} + \mathbf{s} = \mathbf{b}$$

$$\mathbf{y} \cdot \mathbf{A} = \mathbf{c} + \mathbf{t}$$

then we get

$$\mathbf{y} \cdot \mathbf{A} \cdot \mathbf{x} = \mathbf{y} \cdot (\mathbf{b} - \mathbf{s}) = DF(\mathbf{y}) - \mathbf{y} \cdot \mathbf{s}$$

and also

$$\mathbf{y} \cdot A \cdot \mathbf{x} = (\mathbf{c} + \mathbf{t}) \cdot \mathbf{x} = F(\mathbf{x}) + \mathbf{t} \cdot \mathbf{x}$$

so that we have

$$DF(\mathbf{y}) = F(\mathbf{x}) + \mathbf{t} \cdot \mathbf{x} + \mathbf{y} \cdot \mathbf{s}.$$

This equation says that, if both $\mathbf{x}$ and $\mathbf{y}$ are feasible, so that all of the components of the vectors $\mathbf{x}$, $\mathbf{t}$ and $\mathbf{y}$ are positive, then DF, the dual objective function, is always at least as big as F, the primal objective function.

It also implies what we have just finished saying, that in order to obtain equality, one of each pair of variables must vanish, so that $\mathbf{t} \cdot \mathbf{x}$ and $\mathbf{x} \cdot \mathbf{s}$ both vanish, when $\mathbf{x}$ is the primal solution and $\mathbf{y}$ the dual solution.

This duality has a number of uses. The bound it gives (*any dual feasible point gives an upper bound for the primal*) can be useful. Sometimes it is easier to deal with the dual than with the primal; there are approaches to the problem that switch between the two problems. The theorem also has applications in relating seemingly quite different optimization problems.

6 Two-Person Games and Competition in Economics.

We now look at an application of linear programming. Games of all kinds are small-scale and artificial models of reality. Children love to play them, in part because games give them the opportunity to practice and develop skills without being inhibited by the pressures of the real world and the real consequences of actions in it. Games usually have well-defined rules that are abstract and formal (more so than reality), and they end, while reality goes on.

We are particularly concerned with games that model economic behavior, and in particular with games that model decision-making. Studying them and even playing them can shed light on economics and economic decision-making. We consider the role of competition and co-operation in economics, and try to understand it from the point of view of game theory.

Games have been studied in great detail for centuries. However "game theory" as a model for economic behavior was introduced only in the 1930's by John von Neumann. He introduced the notion of a *two-person game* (and an n-person game as well), in which two players simultaneously decide among various alternatives. The outcome of the game depends upon which pair of alternatives is chosen, and we can give a numerical value to the merit of each outcome to each player, called the *payoff*.

Games of this kind can be useful models of many competitive situations. Consider the following example:

Example. This example is typical of a wide class of business decision problems. A and B are managers of competing business ventures. Each discovers an innovation in business practice that can improve the quality of the service or products that this business provides. Should A or B invest in the effort necessary to introduce this improvement or not? Questions of this kind arise very often in every kind of business.

An important class of two-person games, the class that has received the most attention, are those for which the payoff of the game to the first player is exactly the opposite (or negative) of the payoff to the second player. These are called *zero-sum two-person games.* Actually, what is significant is not that the payoffs sum to zero, but that they sum to some constant. Without loss of generality, we can subtract this constant from the payoff to the second player, so that the payoffs indeed sum to zero. Of course, if all the payoffs to a player are negative it seems likely that he will not want to play our game, but we will assume that he is not given a choice!

Zero-sum two-person games have the following property which we shall prove later; there is a strategy for the first player (call her A) and a number v, such that no matter what the second player (call him B) does, A has an expected gain of at least v in playing this game, and there is, similarly, a strategy for B that guarantees B an expected loss of at most v. Moreover, we can find such strategies (and the value v) by solving a linear program.

We remark, however, that this result implies that such games shed little light on the nature of competition, as opposed to cooperation. The two players need not even play the game: B can merely give an amount v to A and this would have the same expected outcome as actually playing out the game. If the players actually play (compete), rather than settle (cooperate), the *expected* payoff to each player is unaffected; the only difference is that there is some risk involved. (Putting it differently: if B decides to hand v to A instead of playing the game, the variance of the random variable "payoff to player B" goes down to zero, but its expected value remains $-v$.) We will see later that in two-person games which are not zero-sum, one can get into "Prisoner's dilemma" type situations, where if players cooperate, they can do better than if they compete.

6.1 Two-Person Zero-Sum Games

A Two-Person Zero-Sum Game may be characterized by giving, for each possible pair of choices for the players, the value of the corresponding outcome to player A. We assume, by convention, that higher values represent more favorable outcomes to A; if this were a gambling game, for example, the value for A might be the number of dollars she has won (a value of -5 would mean that she owes B 5 dollars). We can write all these values in tabular form, as a matrix M where the entry $M(i, j)$ is the value to A when A makes choice i and B makes choice j. (The value to B in this case is of course $-M(i, j)$ since the game is zero-sum.)

The simplest real game of this kind is "choosing odds or evens". There are two possible choices for each player (holding out one or two fingers), and hence four possible outcomes. If the sum of all the fingers held out is even, then the "evens" player wins, and vice versa. Suppose A has "evens". Then we can describe this game by the following 2-by-2 matrix:

$$M = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}.$$

It is easy to see that the optimal strategy for A or B calls for playing each of the alternatives half the time, in as unpredictable a manner as possible. This game is not often played for fun, but as an alternative to tossing a coin, when two people find it necessary to make a fair choice between them.

The game "Scissors, Paper, Rock" is similar; each player has three choices. The nominal value of the game is 0, 1, -1 in the first row, with cyclic shifts to the right of this row in the other rows (where each row and column is assigned the appropriate object; scissor cuts paper, paper covers rock, rock blunts scissors). Again, the game is obviously symmetric among alternatives, and the best strategy is obvious.

$$M = \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}$$

Exercise. Suppose that A, to ensure that her choices are random, uses a three-sided die to choose whether to say "paper," "rock," or "scissors". Suppose, however, that this die is biased, with the respective probabilities p_p, p_r, and p_s not all equaling $\frac{1}{3}$. Finally, suppose that B knows p_p, p_r, p_s. What should B do to *optimize his chance of winning?* What are his chances of winning, and his expected payoff, if he does this? What should B do to *maximize his expected payoff?* What are his chances of winning, and his expected payoff, if he does this? Note that the answers to the two questions are not automatically the same.

There is also an Italian game, Morra, with the same general structure. Players play it rapidly over and over again. Each play involves throwing out some number of fingers on one hand, and simultaneously calling out a number. If A's called number is the sum of the number of fingers on both players' hands, then A scores a point, and likewise for B. Morra is played for pleasure as a gambling game. Skilled players A can analyze the pattern of choices made by B, and anticipate the next choice, giving A an edge. The game is played in part for the joy of taking a risk, and in part to prove and improve the players' skills in anticipating opponents.

In the real world the "simultaneous play" aspect of all these games is not contrived as it is in these examples. Real decisions usually take a long time for their consequences to appear, and in a competitive game-like situation, the decisions have been made by all parties long before the consequences occur. There is no distortion in assuming the players move simultaneously when in practice they may act at different times, so long as each competitor is in ignorance of the other's decision when making his or her own.

There are two kinds of strategies in these games. The simplest, called a *pure strategy*, is to choose an alternative, say the jth, and always play it. A more general kind of strategy is to associate a probability $p(j)$ to the jth alternative, and choose the choice j in as unpredictable a manner as possible, with probability $p(j)$. If two or more $p(j)$'s are non-zero this is called a *mixed strategy*.

If player A uses a strategy defined by $p(j)$ and B makes his kth choice, then the expected return of the game, to A, is

$$\sum_j M(j,k)p(j).$$

Let v denote the value of the game: that is, the maximum return of the game as the strategy $p()$ varies, assuming B plays best possible. Since B can make his kth choice, B can hold A to doing no better than the value $\sum_j M(j,k)p(j)$. So for every k, we have

$$v \leq \sum_j M(j,k)p(j). \tag{3}$$

This gives a bound on v. We also have the conditions that the $p(j)$'s in A's strategy indeed represent probabilities:

$$p(j) \geq 0 \text{ for all } j, \text{ and} \tag{4}$$

$$\sum_j p(j) = 1. \tag{5}$$

These last three equations (3)–(5) are linear constraints on the variables $p(j)$ and v. The *Minimax* strategy for A, is the strategy (or choice of $p(j)$'s) that maximizes v, subject to these constraints. This defines a linear program.

An observation. Suppose that for every one of B's alternatives k, we have

$$M(j,k) \geq M(r,k).$$

This means, no matter what B does, A gets a better result by choosing alternative j than by choosing alternative r. In this circumstance, A will always do better by preferring j to r, and never choosing r if she wants to optimize her result.

There is a natural interpretation of the dual. We have so far considered the problem of finding the optimal strategy only from A's point of view. If we consider B trying to play his best minimax strategy, then this will also give a linear program. If we let $q(k)$ be B's probability distribution over alternatives, and let $-w$ be the value of the game to B (remembering that zero-sum implies that the value of the choices j,k to B is $-M(j,k)$), we find the constraints:

$$-w \leq \sum_k -M(j,k)q(k) \text{ for each } j,$$

and therefore

$$w \geq \sum_k M(j,k)q(k) \text{ for every } j. \tag{6}$$

We also have the constraints that the $q(k)$'s represent a probability:

$$q(k) \geq 0, \text{ for all } k, \tag{7}$$

$$\sum_k q(k) = 1. \tag{8}$$

This linear program "Minimize w subject to constraints (6)–(8) on variables w and $q(k)$" *is the dual linear program to the primal linear program defined by (3)—(5) and "Maximize v ... "!*

What the duality theorem therefore tells us, is $w = v$, that B can expect to lose w by playing minimax strategy, and A can expect to win w, as claimed above.

Exercise. Write down the general two-alternative game for each player and solve it to obtain the best strategy for each, in that each strategy guarantees the best outcome assuming the opponent knows the strategy used, and does best against it, without an ability to predict precise choices.

6.2 The Prisoner's Dilemma

If we relax the restriction of zero-sum, so that one player's loss is not valued exactly at what the other player wins, then the problem becomes much more interesting. To describe a game which is not zero sum, we need two matrices, A and B, whose elements a_{jk} and b_{jk} represent the outcome value to A and to B respectively corresponding to choices j and k. Some games of this kind have properties just like those we have noted for zero-sum games, but some are quite different.

One of those that are different is the game called "The Prisoner's Dilemma." In this game, each player represents a prisoner. A and B have been engaged in criminal activity together, and are taken to the police. If both remain silent, then they have some chance of fooling the police and avoiding punishment; without loss of generality, we give a value of zero to both A and B if this happens. If either player talks to the police, and the other does not, the talker is rewarded, getting a $+1$, while "the book" is thrown at the non-talker, who gets -5. On the other hand, if both talk, then both have less chance of acquittal than if neither talked; they each get an outcome worth -3 compared to 0 for their silent alternative. This game is therefore represented by the matrices:

	B is silent	B talks
A's payoff when A is silent	0	-3
A's payoff when A talks	1	-1
B's payoff when A is silent	0	$+1$
B's payoff when A talks	-3	-1

Notice that the first (or silent) row of the top matrix, which represents the rewards of the row player (A), is term-by-term less than the second row. Therefore, the row player should rationally always choose the "talk" alternative, which is better for A than silence no matter what B does.

Similarly, B's payoff matrix's second column is term-by-term greater than the first, so B is best off talking no matter what A does. Thus simple prudence by the two players separately leads them each to the talking alternative, which gives them each a value of -1. But notice: *This result is worse for both of them than what they would get if they both remained silent.*

This game then illustrates a real difference between cooperative and competitive behavior: rational individual competitive play leads to talking, and a worse result than cooperative play, or silence for each. If the players prudently pursue their own individual interests, they will each talk, and act against their common interest, which is total silence. Have we therefore shown that cooperative behavior is therefore truly required to optimize in some situations?

Cooperation here is indeed the better policy for the two players, but underlying this game is a third player, who actually wins what the two players lose in this game, and loses what they win. The competitive solution is much better for this third player.

Who is it? In the prisoner's dilemma it is the justice system, or the police. What the prisoners gain by cooperation, is the withholding of information from the authorities questioning them. What they give up by competing, they explicitly give up to the questioners (the police). From the point of view of the police, the prisoner's dilemma is a wonderful thing. It means that if prisoners can be persuaded to act in their own individual interest, it is completely rational for them to confess, without any torture, or pressure, or third degree. (It also means for us, that if we view this as a game between three competing players, we no longer have a situation where the best strategy for all players can only be achieved through cooperation.)

Games with this same structure occur in many economic contexts, and give rise to the same general phenomena. With competition, economic players tend to hurt themselves, while cooperation makes life easier for them (at least in the short run). As in the prisoner's dilemma, though, there are again beneficiaries of the loss that competitors inflict on themselves (usually the customers of the businesses); consumers and society (in place of the police) reap the benefits that the competitors lose. Also, in the long run, keen competition forces its participants to become efficient, which allows them to expand to new markets, and ultimately is beneficial to them if they survive. Even for prisoners, punishment and repentance may lead to rehabilitation and ultimate benefit.

When there is cooperation, as is generally the case when governments make the decisions, the decision-makers generally make the choice of convenience to themselves. The silent players, consumers and society, do not benefit.

Consider the first economic game mentioned in the beginning of this section. Suppose, for example, two competing manufacturers each can either:

1. Leave their products alone, or,

2. Make efforts to improve them, in value cost or service.

We again compare outcomes to the null situation, in which neither player decides to act, to which we give the value 0. If one player innovates and the other does not, that is best for the innovator and worst for the player who does nothing. If both players innovate, then both suffer the costs of their innovations, without either gaining market share. As an exercise, the reader should make up some plausible numerical "values" for each of these situations and write down the matrices for A and B. Argue that in a competitive situation, prudence dictates that both players will decide to innovate, while in a cooperative situation the rational solution for the managers will be: don't innovate. You, as manager, and your enterprise can only lose, and cannot win, by innovation and improvement in service in the time frame in which your rewards are determined.

This economic prisoner's dilemma game provides an example of a situation in which competition drives competitors to improve their product or service in the interest of the consumer, and against the interests of their "class". Competition provides a discipline that produces a behavior that does not occur naturally without it. Prisoner's dilemma-type situations suggest a real difference between competitive and cooperative organization in ability to innovate and serve customers.

7 The Buying Problem and Sequential Choice.

Suppose we have to choose one item from among a number of possibilities that are presented to us sequentially. Upon seeing each one, we must decide whether to choose it, or give it up and choose among those as yet unseen. What strategy should we use under these circumstances?

This general problem models many choices in real life and in the economic world:

- We want to make an investment. Should we do it today, or wait for tomorrow. Today's opportunities and prices are gone by tomorrow.

- We want to rent an apartment. If we don't commit to the one we are looking at, it will probably be unavailable by the time we see the next. We had best assume that a passed choice is unavailable (particularly if we are looking in September in Cambridge!), so that this model applies.

- You are hiring someone for a certain job when jobs are easy to come by. Again, the best prospect might take other employment if you delay hiring the present candidate to check out the next one.

- Is X the right prospective spouse for you? Again, you must make your choice without knowing who you may meet tomorrow.

In all cases, we assume that there is a numerical measure of merit that we can associate with each choice as we see it. We also assume that we have no idea what the distribution of this measure is among our potential choices; we assume that the order in which the choices are presented to us is random, for want of any better idea.

7.1 First Problem.

The simplest problem of this kind is when we know our list has length n, and our goal is to pick the very best one, namely the one whose measure of merit is top among all N possible choices. This is often a simplification, since the number of choices is generally somewhat stretchable and not absolutely fixed; also because we are often satisfied with a good choice, rather than the very best. Furthermore, in many real world scenarios, we might be able to use our powers of observation to rank the choices we see against choices that might have been available historically. Thus we might be able to recognize a good choice without looking at any of the others. What we will see now however, is that even when we make *no* assumption about distribution, and our goal is to pick the absolutely best candidate on the spot, we can do surprisingly well.

Observe that since we are looking for the "best" candidate, we obviously must reject any choice that is not the best among those seen so far. Thus the only relevant problem to us is:

If the kth of our alternatives is the best among those seen to date, should we choose it, or pass it and go on?

One thing we could do is simply choose the first candidate, and look no farther. Or we could let three or four candidates go by, and then choose the first candidate who is better than all those already seen. Or we could let all but a few of the candidates go by, and then choose the first candidate who is better than all those already seen, if there is such a one. The more candidates we let go by, the more "informed" our choice is, but also the more likely that we have already passed over and not chosen the best. Clearly, therefore, we seek the *threshold* we wish to find the k_0 for which the strategy "look at the first k candidates, and pass them by; then take the next one which is better than all its predecessors" is most likely to yield us the best candidate. We want to pick k so as to maximize the probability we pick the best choice among all N possibilities. Here is the analysis:

If the best is among the first k, we will not find it, and we will end up with the last choice available. This bad result has, with our assumptions, probability k/N.

Otherwise, if the second best is among the passed over first k, we will definitely stop at the best choice which is only one better than those seen. This good result has probability = (probability that the second best is among the first k choices) times (probability that the very best is not) = $(N-k)(k)/(N(N-1))$.

Otherwise, if the third best is among the first k, we have probability $1/2$ of getting the best (if the best comes before the second best; at either of them we will stop), and our chance of getting the best in this case is then: $(N-k)_{(2)}(k)/(2N_{(3)})$. (Here we are using the standard combinatorial notation $x_{(j)} = x(x-1)(x-2)(x-(j+1))$.)

Continuing in this way, if the $(j+1)$st best is among the first k, but no better candidates are among these first choices, an event with probability $(N-k)_{(j)}(k)/(N_{(j+1)})$, we have a chance $1/j$ of ending up with the best choice, so this contributes the probability $(N-k)_{(j)}(k)/(jN_{(j+1)})$ that we get the best choice in this way.

Our total probability of getting the best choice is therefore

$$P(\text{we make the best choice}) = \sum_{j=1}^{N-k} (N-k)_{(j)}(k)/(jN_{(j+1)}).$$

It is easy to compute this sum for any N and k on the computer, and so obtain the maximizing value of k for each N. If we let $x = (N-k)/N$, then for large N, the problem approaches maximizing the sum:

$$S = (1-x)\sum_{j=1}^{\infty} x^j/j,$$

which is $-(1-x)\log(1-x)$. By differentiating, we find that this function has its maximum when $1-x = 1/e$, and the value of that maximum is $1/e$.

Approximating $1/e$ very roughly, one finds that the best strategy for this problem is to let somewhat more than a third of the possibilities go by, and then take the first choice which is best so far.

7.2 The Second Problem.

The result of the first problem is somewhat misleading, since in most applications the harm done on failure depends heavily on exactly how we fail. If we fail because the best is among the first k, we will end up stuck with the last choice, which we can expect to be only average in merit. If we fail because the best j are not among the first k while the $j+1$st is, we will end up with one of the first j, which is a much better prospect. This suggests that a much more realistic situation, in which we worried more about our expected or average result (worrying about making sure we get a "good" choice, rather than just maximizing the probability we got the best), we should not wait quite so long before making a choice.

Our second problem is that of attempting to minimize the expected rank of our choice among the N possibilities, where the rank of the best is 1, second best is 2, etc. In this problem **we should accept a choice revealed to us if its expected rank, based on what we know of the past (i.e. based on its rank among its predecessors), is less (i.e. better) than the rank we can expect from our future candidates if we pass it.**

This condition defines a recurrence relation that we can solve to determine the best strategy and the expected or average rank we will achieve with this strategy.

We first ask the question; if the kth choice has rank j among the first k, what is its expected rank among all N choices? The answer is $j(N+1)/(k+1)$, which we now prove:

Theorem 2 *(a) If the rank of a particular choice C among k alternatives is j, then if the population is expanded by 1, and the new choice has equal probability of having any rank from 1 to $k+1$, the expected rank of C among the $k+1$ alternatives is*

$$E(\text{rank of } C) = j(k+2)/(k+1).$$

(b) If the expected *rank of a particular choice C among k alternatives is j, then if the population is expanded by 1, and the new choice has equal probability of having any rank from 1 to $k+1$, the expected rank of C among the $k+1$ alternatives is*

$$E(\text{rank of } C) = j(k+2)/(k+1).$$

(c) If a total of $N-k$ additional choices are added sequentially in the same way, the expected rank of C is

$$E(\text{rank of } C) = j(N+1)/(k+1).$$

Proof. If our particular choice has rank exactly j among the first k choices, there are $k+1$ possible ranks for the added choice, and of these j will be below the rank of our choice, and hence will increase its rank by 1. The expected rank of our choice will therefore increase by $j/(k+1)$, from j to $j+(j/k+1)$, that is to $j(k+2)/(k+1)$. This proves (a).

Since this result is linear in j, and the expectation of a constant multiple of random variable is the same multiple of the expectation of the variable itself, this same claim holds if the expected rank of our choice is j, which proves (b).

If the additional $N-k$ choices are added in sequence, repetitive application of (b) imply (c). □

Our strategy for the second problem will be as follows. Suppose we look at our kth candidate and determine that it has rank j among the candidates already seen. By our theorem, its expected rank in the whole population is $(N+1)j/(k+1)$. We choose this candidate if its number is less than the rank we can expect to obtain if we pass it.

In order to implement this strategy we must determine this last quantity. Let $r(N, N-k)$ be the rank that we can expect if we follow our strategy after rejecting the kth candidate, leaving us with $N-k$ as yet unseen candidates. Let $j(N,k)$ be the threshold, or largest rank among its predecessors for the kth candidate which would cause us to choose it. We then decide to choose the kth candidate whenever $(N+1)j/(k+1) < r(N, N-k)$, and hence have

$$j(N,k) = \lceil (k+1)r(N, N-k)/(N+1) \rceil.$$

or

$$j(N, N-k) = \lceil (N-k+1)r(N,k)/(N+1) \rceil. \tag{9}$$

We will compute $r(N,k)$ recursively as follows.

$r(N,1)$ is the expected rank if we have only one remaining choice and must choose it. By our assumptions, $r(N,1) = (N+1)/2$, the average rank, over all possible equally probable ranks, $1, \ldots, N$. This by equation (9) tells us that $j(N, N-1) = \lceil N/2 \rceil$.

We can compute $r(N,2)$ from $r(N,1)$ and $j(N, N-1)$. If the rank of the $(N-1)$st candidate among its predecessors is $j(N, N-1)$ or less, we will choose it, and hence obtain average rank of $(j(N, N-1)+1)/2$. This would happen for $j(N, N-1)$ of the possible ranks for the $(N-1)$st candidate out of the $N-1$ possibilities. Otherwise, we will pass this candidate and get expected rank $r(N,1)$. We therefore compute $r(N,2)$ by the equation:

$$\begin{aligned} r(N,2) &= (\text{ prob. of passing up choice } N-1\ ) * r(N,1)\ + \\ &\quad\ (\text{ prob. of choosing } N-1\ ) * (\text{ expected rank if we do })\end{aligned}$$

$$\begin{aligned} r(N,2) &= (1 - j(N, N-1)/(N-1)) * r(N,1)\ + \\ &\quad\ j(N, N-1) * (N+1)(j(N, N-1)+1)/(2N(N-1)).\end{aligned}$$

By identical reasoning, we may deduce that

$$\begin{aligned} r(N, k+1) &= (1 - j(N, N-k)/(N-k)) * r(N,k) + \\ &\quad\ j(N, N-k) * (N+1)(j(N, N-k)+1)/(2(N-k)(N-k+1)).\end{aligned} \tag{10}$$

The pair of recursions (9), (10) determine the functions $j(N,k)$ and $r(N,k)$ completely, and we may therefore compute $r(N,N)$, the rank we can "expect" to reach from the start.

In the old days, you would have to take some further steps to make anything of these results. Now on the meagerest computer we can work out the values of these functions for any values under a million almost instantly. In order to develop some intuition about the nature of the best strategy for this function, it is necessary either to do analysis or evaluate these functions for enough values to get a feeling for their implications. We can deduce the average number choices needed under this strategy by reasoning similar to that used to derive the recursions. Here are some questions worth answering.

1. Can one describe the optimal strategy implied by (9), (10) by some simple "rule of thumb" that would allow us to use the strategy in practice without remembering much or resorting to lots of computation?

2. What can we achieve in terms of expected rank by this strategy?

3. What happens in the limit as N increases to infinity?

Some analysis has been performed on this last problem by Herbert Robbins, and collaborators. By evaluating the functions one can observe the following qualitative empirical facts:

1. The best strategy is roughly as follows: let a quarter of the choices go by. Among a quarter of what is left use $j = 1$, that is choose only any that is the best so far; among a quarter of what is left use $j = 2$, a quarter of what is left $j = 3$, etc.

2. With the best strategy, the expected rank is somewhat below 4, even as N goes to infinity. That is, on average, the rank of your choice will be better than fourth, among all N possibilities, no matter how large N is.

We included some empirical computed results below.

7.3 Comments

All of this suggests that certain amount of patience is prudent in making sequential choices of this kind. In the real world, of course, there are costs to delaying choice in money and in wasted time that we have not taken into consideration at all. It is possible to modify our analysis by putting in a penalty against passing to take account of this cost.

We can, for example, introduce a penalty of rank x for each choice we pass, and therefore use as a criterion for passing, that the expected rank if we choose is no more than x greater than the expected rank if we pass. The only change in our recurrences is that (9) must be replaced by

$$j(N, N-k) = \lceil (N+1-k)(r(N,k)+x)/(N+1) \rceil. \tag{11}$$

Another feature of real world situations is that there is rarely an absolute constant N that determines exactly how many possibilities we have; typically we would like to finish our search as soon as possible, and we have in mind a number, M, that represents the number of choices we would be comfortable examining in the process. It is then possible to adjust the parameter N so that the expected number of choices seen in problem 2 is near M, and use that value of N with the strategy determined in problem 2.

There are many variations of this problem, some of which are highly relevant to real world decision-making. Many can be explored quite easily through computer solution of the recursions.

It is prudent to keep this problem in mind, along with the results we obtain for it, when you do some real decision-making: if we use this strategy, the expected number of choices we actually look at is something like $.64N$. If you use it, and feel it appropriate to look at 10 choices, you could set $N = 15$ and use the strategy; this will yield a choice expected to be at rank 3 among 15, which s at the upper 20th percentile; if you were willing to look at 64 choices, you could set $N = 100$, and expect to get a result at the upper 4th percentile.

Exercises.

1. Write a program that computes $r(N,k)$ and $j(N,k)$ for all k and any N, and compute all values for $N = 20, N = 40, N = 80$.

2. Figure out how to compute the expected number of candidates to be seen; if we let $g(N,k)$ be the number of choices that we expect to make given that we have passed, and there are k candidates left, determine a recursion for $g(N,k)$.

3. Determine the behavior of the expected number of candidates seen under our strategy as a function of N.

7.4 The Stock Market Problem

In investing in the stock market, one should usually make buy-or-sell decisions based on long-term expectations, rather than on considerations of price fluctuations. That being understood, one can still attempt to time one's actual trades based on considerations like those used here. In the problems described so far, we have been trying to do two things at once; make our choice and learn about available choices, in situations in which we have no advanced information on what is available.

In the case of stock market purchases, we have considerable information about the past which may or may not be useful for the learning task. At issue is, what does this history imply about future choices? We assume first that over periods of say 20 business days, closing stock price occur in random order. Then if we wish to buy or sell a given stock sometime in the next 20 days, we could use the $N = 20$ version of our strategy to decide when to buy. Thus we can wait a few days and then for a few days buy the first time the price goes below its lowest close on those days, etc.

Since information is available about past closing prices, we can even use the $n = 20$ version of the strategy, starting 5 days ago. That is, we can use past information to inform us of the current state of the 'trading range' for the stock, and buy when the price appears to be low in comparison with that range, thus eliminating the purely learning stage of the choice process.

This represents a first order strategy for buying or selling securities. An interesting question is whether it is a wise one. Actual stock closing prices exhibit random fluctuations, but they also undergo long-term trends. Stocks, furthermore sometimes move roughly in the same direction for a period of time, so that our assumption of ranks uniformly distributed in time may not hold.

It may be wisest to realize that many people do use strategies akin to ours; they determine what they think is a reasonable trading range, buy the stock when it is low in that range, and sell it when it is high in it. Such activities tend to keep the price within that range.

This suggests that a trend in stock is significant when a stock price breaks out of its trading range and, for example, makes and maintains a new high. Such a high often means that there is buying pressure strong enough to overcome normal selling at the top of the trading range. Such pressure often has some real significance. A second order strategy might therefore be: look for stocks in the long term that have just made new highs, not new lows.

Exercise. (optional) Choose a stock and compare the results of buying it the closing price at day 1, or buying at any time after day 1 that it goes below the threshold indicated by our strategy based on the closing prices with $N = 20$ starting at day -5. Do the same for selling it.

18.310: Introduction to Discrete Applied Math — Professor Propp
Massachusetts Institute of Technology — Last Revised: September, 1995

Supplementary Notes 2: Duality and The Simplex Method.

(This note is intended to supplement the material in chapter VI.)

We begin with the problem

$$\mathbf{P0}: \begin{array}{l} \text{Maximize } 2x + 3y, \text{ subject to} \\ x + 2y \leq 4, \\ 3x + 2y \leq 8. \end{array}$$

(It is understood that all variables discussed here must be non-negative.) This is a simple enough program that we can sketch the feasible region and solve the problem by inspection; see Figure 1. However, we will use a form of the simplex method instead, to take us from the point $x = y = 0$ to the point $x = 0$, $y = 2$ and then to the point $x = 2$, $y = 1$.

Introducing slack-variables s and t, we rewrite ($\mathbf{P0}$) as

$$\mathbf{P1}: \begin{array}{l} \text{Maximize } 2x + 3y, \text{ subject to} \\ x + 2y + s = 4, \\ 3x + 2y + t = 8. \end{array}$$

Figure 3: The (primal) linear program.

We start at the point $x = y = 0$. Note that this is a vertex of the feasible region. If we increase x while holding y at zero, and want the equality constraints to remain satisfied, we will need to have $s = 4 - x$ and $t = 8 - 3x$. Since s and t are required to stay non-negative, x can increase to $\frac{8}{3}$ but no further. (This leaves s positive but makes t zero; any further increase in x would make t negative.) Putting $x = \frac{8}{3}$ and $y = 0$, we get $2x + 3y = \frac{16}{3}$. On the other hand, if (starting at $x = y = 0$) we increase y while holding x at zero, we can increase y to 2, making s zero and raising the objective function $2x + 3y$ all the way to 6. 6 is greater than $\frac{16}{3}$, so we choose to move to the latter vertex. (Note that we are using the option labelled (d) in section 2.2, remark #1: always to go to the neighboring vertex at which the objective function is as large as possible. If we had opted for (c), we would have made the same choice between the two neighbors of the origin, but for a different reason: the coefficient of y in $2x + 3y$ is greater than the coefficient of x.)

Now we have $x = s = 0$; we modify the program so that y and t play the role of slack variables. That is, we rewrite (**P1**) so that y occurs in just one equation (with coefficient $+1$) and t occurs in just the other (also with coefficient $+1$). Dividing the first equation in (**P1**) by 2, we get $\frac{1}{2}x + \frac{1}{2}s + y = 2$. Subtracting the first equation in (**P1**) from the second, we get $2x - s + t = 4$. Finally, using the substitution $y = 2 - \frac{1}{2}x - \frac{1}{2}s$, we find that the objective function $2x + 3y$ can be rewritten in terms of x and s as $2x + 3(2 - \frac{1}{2}x - \frac{1}{2}s) = 6 + \frac{1}{2}x - \frac{3}{2}s$. Thus we have

$$\mathbf{P2}: \quad \begin{array}{l} \text{Maximize } 6 + \frac{1}{2}x - \frac{3}{2}s, \text{ subject to} \\ \frac{1}{2}x + \frac{1}{2}s + y = 2, \\ 2x - s + t = 4. \end{array}$$

We are at the point $x = s = 0$. If we increase x to 2, the objective function increases to 7. If we increase s by any amount, the objective function decreases (since s has a negative coefficient in the objective function). So our only choice is to increase x to 2, thereby making t equal to 0.

Finally, we come to a situation in which x and y are non-zero and must play the role of the slack variables. Rewriting our linear program in terms of s and t, we get

$$\mathbf{P3}: \quad \begin{array}{l} \text{Maximize } 7 - \frac{5}{4}s - \frac{1}{4}t, \text{ subject to} \\ -\frac{1}{2}s + \frac{1}{2}t + x = 2, \\ \frac{3}{4}s - \frac{1}{4}t + y = 1. \end{array}$$

The coefficients of both s and t in the objective function are negative, so increasing these variables will not help us. We have arrived at the maximizing vertex, and the simplex algorithm terminates.

(The reader should not make the mistake of thinking that every linear program has a feasible region that looks like the one shown in Figure 1; we chose (**P0**) the way we did because it allows us to find the optimal vertex in two pivots. The feasible region shown in Figure 2 of Chapter VI shows another sort of feasible region; and in higher dimensions, there are still more possibilities.)

To motivate the notion of a dual program, go back to (**P0**), and consider that one can show right off the bat that the optimal value of the objective function cannot exceed 12. For, by adding together the two constraint-inequalities, we get $4x + 4y \leq 4 + 8 = 12$, and since $2x + 3y \leq 4x + 4y$ (on account of the non-negativity of x and y), we must have $2x + 3y \leq 12$ for any x, y satisfying the stated inequalities. Indeed, this idea generalizes: Given any two non-negative coefficients a and b, we can multiply the first inequality of (**P0**) by a and the second inequality by b, and add, obtaining

$$a(x + 2y) + b(3x + 2y) \leq a \cdot 4 + b \cdot 8$$

or

$$(a + 3b)x + (2a + 2b)y \leq 4a + 8b.$$

This will give $4a + 8b$ as a bound to our original program, provided that $a + 3b \geq 2$ and $2a + 2b \geq 3$. Thus, to get the best (i.e., lowest) possible upper bound on the value of the linear program (**P0**), we seek the smallest possible value of $4a + 8b$, subject to the constraints $a + 3b \geq 2$ and $2a + 2b \geq 3$. This is nothing other than the *dual* of (**P0**). We write it as

$$\mathbf{D0}: \begin{array}{l} \text{Minimize } 4s' + 8t', \text{ subject to} \\ s' + 3t' \geq 2, \\ 2s' + 2t' \geq 3. \end{array}$$

(Here we change the names of our variables from a, b to s', t' to facilitate later mischief. Even now you should be able to see a sort of method to this madness: the variables s' and t' correspond to the two respective inequalities of (**P0**), whose slack variables were s and t.) The dual program is sketched in Figure 2.

Note that every solution to the dual program gives an upper bound on all possible solutions to the original (or *primal*) program. Equivalently: Every solution to the dual program is greater than or equal to every solution to the primal program. (We are abusing language here: we should really be speaking of the values of the objective function associated with a solution, not the solution itself.) In particular, the optimal (minimal) solution to the dual program is greater than or equal to the optimal (maximal) solution to the primal program. The duality theorem asserts that in fact, *equality* holds between the two optimal solutions; there is no gap between the values of the primal objective function achieved by feasible solutions to the primal program and the values of the dual objective function achieved by feasible solutions to the dual program.

Figure 4: The dual linear program.

The simplex method can be a conceptual tool in understanding the duality theorem. In particular, it is helpful to imagine that when one is applying the simplex method to the primal, one is also applying it to the dual. However, there is an asymmetry to the situation. For the primal problem, one starts with a point in the feasible region and moves it towards optimality; for the dual problem, one starts with a point outside of the feasible region and moves it towards feasibility.

Let us rewrite (**D0**) so that it, like (**P1**), involves slack variables. Since the two equations of (**P1**) correspond to the two variables x and y of the primal, we call the new slack variables x' and y', respectively. Thus we obtain

$$\mathbf{D1}:\quad \begin{array}{l} \text{Minimize } 4s' + 8t', \text{ subject to} \\ s' + 3t' - x' = 2, \\ 2s' + 2t' - y' = 3. \end{array}$$

(Note that the dual slack variables appear with minus signs, since the inequalities in this program go the other way.) We need a starting point for this program; we take $s' = t' = 0$. It is not a feasible point, since it gives negative values to x' and y'. To move towards the feasible region, let us increase s' or t' so that (treating x' and y' for the moment as dependent variables) either x' or y' will cease to be negative. If we increase s' to $\frac{3}{2}$, y' will become zero; if we increase t' to $\frac{2}{3}$, x' will become zero. Which do we choose? Let us compare the values of the dual objective function corresponding to these two new vertices, and choose the one which gives the *greatest* increase. Thus, since $s' = \frac{3}{2}$, $t' = 0$ gives $4s' + 8t' = 6$ while $s' = 0$, $t' = \frac{2}{3}$ gives $4s' + 8t' = \frac{16}{3} < 6$, we are instructed to opt for the former.

Now we have $t' = y' = 0$, so we ought to regard x' and s' as dual slack variables and rewrite (**D1**) accordingly. Easy algebra gives

$$\mathbf{D2}: \begin{array}{l} \text{Minimize } 6 + 2y' + 4t', \text{ subject to} \\ \frac{1}{2}y' + 2t' - x' = \frac{1}{2}, \\ \frac{1}{2}y' - t' - s' = -\frac{3}{2}. \end{array}$$

To finish, we increase t' to $\frac{1}{4}$. The dual program becomes

$$\mathbf{D3}: \begin{array}{l} \text{Minimize } 7 + 2x' + y', \text{ subject to} \\ -\frac{1}{2}x' + \frac{3}{4}y' - s' = -\frac{5}{4}, \\ \frac{1}{2}x' - \frac{1}{4}y' - t' = -\frac{1}{4}. \end{array}$$

Note that $x' = y' = 0$ is a feasible solution to this program; since this is as small as x' and y' can be (remember, all variables are non-negative), this feasible solution to the dual program is also optimal.

You may at this point be struck by a general sense of déja vu, and specific feeling that you've seen the numbers $\frac{16}{3}$, 6 and 7 before. If so, look back at our solution to the primal, and notice that our solution to the dual involves the same numbers at each step, playing different roles but in a consistent manner. More specifically, the programs (**D1**), (**D2**), and (**D3**) are dual to the respective programs (**P1**), (**P2**), and (**P3**). (To see this, drop the slack variables and replace the equalities by inequalities.) Also note that there is a relation between the vertices we visited in solving the primal problem and the (mostly non-feasible) vertices we visited in solving the dual program. Specifically, we went from $(x, y) = (0, 0)$ to $(x, s) = (0, 0)$ to $(s, t) = 0$ in the primal, and from $(s', t') = (0, 0)$ to $(y', t') = (0, 0)$ to $(x', y') = 0$ in the dual. The correspondence is simple: the non-slack variables in the dual program are just the primed versions of the *slack* variables of the primal.

Note that the optimal solution to the dual is $s' = \frac{5}{4}$, $t' = \frac{1}{4}$. There's no mystery here: setting $a = \frac{5}{4}$ and $b = \frac{1}{4}$ (where a and b are the coefficients we discussed in our first mention of the dual program), we can check that by adding $\frac{5}{4}$ of the first inequality in (**P0**) to $\frac{1}{4}$ of the second inequality in (**P0**), we get $2x + 3y \geq 7$. Coupling this with the fact that $2x + 3y$ *can* be 7 in the primal (witness the point $x = 2$, $y = 1$), we see that 7 is indeed the value of the primal program. Also, recalling that $x' = y' = 0$ was a feasible as well as optimal solution to (**D3**), we see that 7 is also the value of the dual program.

Finally, let us use the insight we have gained from the study of this example to figure out why it should be that, when we perform the simplex algorithm simultaneously on the primal and dual programs, we find a feasible solution to the dual program at the same time as we hit the optimal solution to the primal program. Since we have set things up so that the current vertex is always represented by the origin of some coordinate system, it suffices to show that the origin $(0, 0, ..., 0)$ is an optimal solution to the primal program if and only if the dual origin $(0, 0, ..., 0)$ (obtained by setting equal to zero the primed versions of those variables that have *not* been set equal to zero in the primal) is a feasible solution to the dual program. The all-zeroes vector is *optimal* for the primal problem if no further progress can be made by increasing the non-slack variables; that is, all the coefficients in the primal objective function have become negative (aside from the constant term). The all-zeroes vector is *feasible* for the dual problem if all the constraints are satisfied; that is, the entries in the constraint-vector for the dual program are all less than or equal to zero. But recall that the coefficients of the primal objective function are precisely the entries of the constraint-vector for the dual! Hence, a vertex is optimal for the primal problem exactly when the dual vertex is feasible for the dual problem.

(Note that if our primal program involves n variables and m constraints, then our dual program involves m variables and n constraints. Thus, the feasible regions are in general n- and m-dimensional, respectively, and do not at all resemble each other. It was fortuitous that in the example we chose, m and n were equal to one another, so that Figures 1 and 2 looked misleadingly similar.)

Now we can understand why the duality theorem holds: When the simplex algorithm on the primal side hits an optimal solution (which is feasible), the simplex algorithm on the dual side hits a feasible solution (which happens to be optimal for the dual program, but never mind that). Thus, we get a primal solution and a dual solution which yield the *same value* v for their respective objective functions. (Remember, at each stage of our joint simplex algorithm, the value of the primal objective function equals the value of the dual objective function.) Since v is feasible for the dual program, v must be an upper bound on the optimal solution to the primal. But v is indeed achieved by the primal! Hence v is optimal.