An Introduction to Advanced Linear Algebra

Steven J. Miller[*]

January 13, 2012

Department of Mathematics and Statistics
Williams University
Williamstown, MA 01267

Abstract

The abstract below describes the content of the notes written so far; before delving into these notes, we first sketch some thoughts about the class and the book.

We describe Linear Programming, an important generalization of Linear Algebra. Linear Programming is used to successfully model numerous real world situations, ranging from scheduling airline routes to shipping oil from refineries to cities to finding inexpensive diets capable of meeting the minimum daily requirements. In many of these problems, the number of variables and constraints are so large that it is not enough to merely to know there is solution; we need some way of finding it (or at least a close approximation to it) in a reasonable amount of time. We describe the types of problems Linear Programming can handle and show how we can solve them using the simplex method. We discuss generalizations to Binary Integer Linear Programming (with an example of a manager of an activity hall), and conclude with an analysis of versatility of Linear Programming and the types of problems and constraints which can be handled linearly, as well as some brief comments about its generalizations (to handle situations with quadratic constraints).

[*]E-mail: `sjm1@williams.edu`

Contents

1 Course / Book Outline

The following is a rough outline of ideas for a course on advanced linear algebra at Mount Holyoke for the Spring 2012 semester, which will serve as a nucleus for a textbook. The style of the book and the course will be conversational and friendly.

1.1 Motivating Questions

Here are four questions I'm thinking about.

1. **Should the course be just linear programming, or random matrix theory too? What about the book?**

   ⋄ There is more flexibility if we do both, more chance of showing students where things can go, but for a book it might be a bit too much (though it would of course also give instructors more flexibility). The advantage of just doing linear programming and not random matrix theory is that we can make this also a brief introduction to linear algebra, and build on that. Thus, for now, the plan for the book is *not* to include much random matrix theory, at most a few pages on it at the end of the book in a 'what's next' section. It would fit well there, as the rest of the book is devoted to studying fixed matrices. In fact, we could even use this as a nice bridge / transition. In the real world, we of course don't know exact values. What happens if we start drawing the entries at random? There's some nice work on this subject, leads to a quick introduction to probability, and then can end with random matrices. This is stochastic linear programming. Lots of references. A quick search yielded the following paper: http://www.jstor.org/pss/1907736; a more extensive review of the literature is needed.

   ⋄ Goal for the course: see where the math goes, use the material as a springboard to introduce topics throughout mathematics. The goals for the book will be very similar, with the big addition of getting comfortable with mathematical modeling (a very important and marketable skill), being able to set up linear programming problems, having some idea of what resources are out there to solve them, et cetera.

2. **Market for the book / course: Is this a general course with minimal pre-reqs, or is this a sequel to calculus?**

   ⋄ My original thought was that, as much as possible, let's minimize calculus. This will increase marketability. We can have calculus asides. After talking with ED, it might be good to include a calculus pre-req. This will make sure students have some mathematical maturity and familiarity. Also, of course, the biggest weakness students have in calculus classes is lack of algebra skills, and that would be very hard in a course such as this.

$\diamond$ We don't need too much calculus for linear programming; at most, maybe some comments on optimizing problems and boundary issues. If we start talking about constraint matrices with random entries, however, then a bit more calculus is needed as we now must discuss probabilities. We wouldn't need too much from probability; many of the concepts students should have already seen. A big advantage of this is that many people do not know that one of the biggest applications of calculus is that areas are probabilities. We can thus motivate, after the fact, the calculus they learned.

$\diamond$ For random matrix theory, we would need more probability. As there are more than enough interesting items in linear programming, delving this deeply (in either the course or the book) seems to be a mistake, and instead we should content ourselves with references to the literature.

3. **What linear algebra is needed?**

$\diamond$ We'll review what is needed below. This is a quick pass – as the semester progresses and we start dealing with the material, we'll have a better sense. We can decide later for a book if this should be an introductory chapter, or an appendix.

$\diamond$ First is $A\vec{x} = \vec{b}$. The left hand side is a linear combination of columns. This is a very important idea for linear programming. We'll talk a lot about linear combinations. This is not just important in linear algebra, but also in differential equations / difference equations (linear combinations of solutions, Binet's formula for Fibonacci numbers). Another example is portfolio theory, where we want to minimize variance subject to a given expectation.

$\diamond$ Have to deal with degenerate cases: $A\vec{x} = \vec{b}$. This is not always solvable, talk about determinants.

$\diamond$ Tweaking answers: probability of an event: 'almost' all matrices with real entries are invertible. Need, however, to worry about numerical stability, and this is a fascinating topic and can make for some nice asides. Maybe talk a bit about computational complexity, such as (my favorites) fast exponentiation (cryptography), Horner's algorithm (fractal geometry), Strassen's algorithm (fast matrix multiplication – huge applications). Just 'show' or briefly describe these possibilities (or maybe these are advanced topics / supplemental topics at the end of the book).

$\diamond$ Talk about the difference between real versus integer solutions. Some riddles on the number of animals satisfying given constraints, and there are infinitely many 'real' solutions, but if you require the solutions to be integers (which makes sense, as no one has e/π of a cow) then there is essentially 'one' solution (or maybe up to some multiples, but we often need to have non-negative numbers). Another example is the optimal base for storing information in a computer; answer turns out to be base e (a very nice calculus problem, uses almost everything you learn in Calculus I), and thus base 3 is surprisingly better than base 2 in this respect (as 3 turns out to be closer to e than 2). This needs naturally into (1) the backpack problem, (2) having to take specific roads (and either go to the city or don't), (3) .... This is a very real problem / complication in the real world; you either do something or you don't, there is no temporizing. No one is half-pregnant; you don't have a flight where some people go to Tucson and some to Denver. You can, however, temporize a bit by breaking flights into legs.... Note that this leads to our current hub system.

$\diamond$ Need some combinatorics, nothing too advanced, mostly on the level of $\binom{N}{M}$, as that arises in determining number of options, which is useful for determining how long something will take.

4. **What is needed for Random Matrix Theory?**

   $\diamond$ Though the book is almost surely going to just be linear programming and what spins off, if we do random matrix theory what is needed? Need some basic probability, the fact that integration gives areas gives probabilities, eigenvalues, and of course combinatorics. To 'prove' the theorems in full detail requires a lot of advanced material, but there is no need to give a full proof; one can argue about the reasonableness of the method of moments via a comparison with Taylor series and call it a day.

1.2 Content for Advanced Linear Algebra

Here are three thoughts.

1. **Why Linear Algebra? Where is it used?**

   $\diamond$ Let's start with some standard linear algebra problems to see where we began. Everyone is familiar with the boring problem of two trains leaving at different times and at different speeds and wondering when they meet. This is the classic example of a linear algebra problem, but doesn't showcase *why* the subject is so important.

   $\diamond$ Perhaps talk about perspective lines in art?

   $\diamond$ Advanced application: cryptography: error correction / detection, such as the Hamming (7,4) code. We don't have to go into the entire theory (or maybe we do this as a supplemental chapter for a longer, more advanced course; similar comments for the next few items).

   $\diamond$ Advanced application: fast Fourier transform, lots of signal processing applications.

   $\diamond$ Advanced application: Method of Least Squares, fundamental in fitting.... I do have extensive notes on this, could make that an appendix if it is worthwhile including (to give a professor some flexibility in making a course).

   $\diamond$ Another possibility is to talk about calculus, and the need to linearize non-linear phenomena. This is probably a good idea as it then builds on the calculus we say we require.

   $\diamond$ Another item is difference equations, where we can write solutions as iterates of a matrix (this ties in with efficiency and many other items).

   $\diamond$ Describe the types of solutions to linear algebra problems: zero, one (unique) or infinitely many.

   $\diamond$ Most things sadly non-linear. It is hard to solve problems exactly. Often we get approximate answers to approximate equations for the real world problem. This is something many people haven't experienced; we're so used in the first few math classes of having clean equations with clean solutions. This isn't the case in the real world. We have two conflicting items: we want our model to accurately describe the real world, and we want it to be mathematically tractable. These are competing goals.

2. **Non-linear Equations**

   $\diamond$ There are many ways to generalize from Linear Algebra. One is to go from linear equations to non-linear equations, which we discuss now.

   $\diamond$ Math can be 'simple' but have lots of applications. A great example is GPS systems (motivated by my son Cam asking how the computer knows where we are). Imagine you have a few satellites that tell you how far they are from you. Each one localizes you to a surface of a sphere, and then you need to find the points of intersection to uniquely determine where you are. This can tie into the Method of Least Squares (get an overdetermined system).

$\diamond$ Advanced application: Hamiltonians from Physics. Other possibilities are planetary trajectories, ballistics, ... Lots of possibilities here!

3. **Linear Programming (and beyond)**

$\diamond$ Instead of generalizing to non-linear equations, we can generalize another way: we now want to optimize a function subject to various constraints. The most general case gives non-linear constraints, which sometimes is doable but is quite hard. We discuss now linear programming.

$\diamond$ Start with optimization with linear constraints. Much of life is optimization. We can talk about Physics (Hamiltonians, Feynman's action, ...), soap bubbles, economics, ...

$\diamond$ Main problem: given $A\vec{x} = \vec{b}$, optimize $\vec{c}^{\mathrm{T}}\vec{x}$. We'll see later that there is no loss of generality in using equality instead of inequalities in the matrix constraints. Also constraints on the variables. Typically take $\vec{x} \geq 0$, and again will see there is no loss. Here we can talk about 'canonical' formulations. This is a big concept in mathematics, namely that it is sufficient to reduce certain general problems to special cases. Examples abound (find them!). One is solving the cubic – enough to consider the depressed cubic. Another nice example is tic-tac-toe; there aren't 9 opening moves, but up to isomorphism just 3.

$\diamond$ In solving linear programming problems, we'll learn about the dual problem. Similar to the argument above (getting a canonical formulation), this is another big concept. It seems strange, but a related problem is often much easier. A great (but high level!) version of this is Poisson Summation; try and find other examples.

$\diamond$ Lots of applications to discuss. Big one is the oil industry (have a bunch of oil sources, a bunch of refineries, a bunch of markets – where is oil shipped from). Another is the airlines (this led to the hub system we have now, figuring out what flights planes should take, but there are lots of problems because we don't know the 'true' demands for people for flying from city A to city B on day D). There are a plethora of scheduling problems (can do movie theaters: what movies to buy, what screens to use, what times to start, but again, we often have to estimate demand and interaction effects, which are fascinating problems in their own right). We can also look at baseball. One possibility is describing a schedule (lots of issues, such as hard constraints, but also wanting to have big games between popular teams at popular times, save division games for the end of the season, ...). Another possibility is the fascinating problem of elimination numbers (some great papers on this, and how the press *miscalculates* this!). One final problem: pure mathematics: Hales proof of the Kepler conjecture (on sphere packing) involved solving hundreds of thousands of linear programming problems.

◇ A key question is how much theory to get into. While the simplex method is not the best algorithm out there, it has a very approachable proof, which is worth doing in full detail. The proof also has a nice outline. It's broken into two parts, Phase I and Phase II (using the notation from Joel Franklin's excellent book – need to check if this is the original notation). First, assuming we can do Phase II we prove we can do Phase I. Then, using Phase I, we prove we can do Phase II. This sounds circular; it's not – to prove Phase I, we only need to know how to do Phase II for a related problem, and in that case it is *trivial* to show Phase II is doable. This is a beautiful idea, and worth grokking fully. Other methods to solve linear programming should be mentioned and referenced, but not really discussed.

◇ Note in the various linear programming problems mentioned, many are Integer Linear Programming problems, or Binary Integer Linear Programming problems. I find formulating these a lot of fun. Specifically, so many relations that do not look linear can be made linear (such as max/min, absolute values, cut-off functions) at the cost of introducing more variables and constraints. This provides a wonderful platform to discuss trade-offs. Would you rather do lots and lots of simple problems, or a few hard problems? Most of the time, it's better to do the simpler problem, especially when we have cranks to churn out the answer. Maybe motivate solving a general polynomial to solving a chain of quadratic equations. There is of course a run-time cost for adding variables and constraints, but linear programming exists, while quadratic and higher is either harder or is not available. The motivating quote should be: *What is amazing is not that the horse sings well, but that the horse sings at all.*

◇ The problems above (except maybe Hales' proof of the Kepler conjecture) involve linear programming problems, where the parameters in the constraint matrix are fixed. In the real world, we won't know these values exactly, and this leads to stochastic linear programming, where these parameters are now drawn from distributions (hopefully known). A little probability is needed here. The goal would probably be to do a few examples to give the flavor, but not proofs in general, just a literature review and an attempt to get things on people's radar.

2 Linear Programming

2.1 Introduction

We describe the ideas and applications of Linear Programming; our presentation is heavily influenced by Joel Franklin's excellent book, *Methods of Mathematical Economics* [Fr]. We strongly recommend this book to anyone interested in a very readable presentation, replete with examples and references.

Linear Programming is a generalization of Linear Algebra. It is capable of handling a variety of problems, ranging from finding schedules for airlines or movies in a theater to distributing oil from refineries to markets. The reason for this great versatility is the ease at which constraints can be incorporated into the model. To see this, in the following section we describe a specific problem in great detail, and in §4 we discuss how some quadratic (or higher order) constraints can be handled as well.

2.2 The Canonical Linear Programming Problem

2.2.1 Statement of the Diet Problem

We consider a simple case of the diet problem first, and then discuss the generalization. Assume for simplicity that there are only two foods, which for definiteness we shall assume are cereal and steak. Further, we assume that there are only two products people need to stay alive, iron and protein; each day a person must consume at least 60 units of iron and at least 70 units of protein to stay alive. Let us assume that one unit of cereal costs \$20 and contains 30 unit of iron and 5 units of protein, and that one unit of steak costs \$2 and contains 15 units of iron and 10 units of protein. The goal is to find the cheapest diet which will satisfy the minimum daily requirements. *We have deliberately chosen absurd prices and levels of iron and protein for cereal and steak to illustrate a point later.*

Let x_1 represent the number of units of cereal that the person consumes a day, and x_2 the number of units of iron consumed. For the diet to meet the minimum requirements, we must have

$$\begin{aligned} 30x_1 + \ 5x_2 &\geq 60 \\ 15x_1 + 10x_2 &\geq 70 \\ x_1 &\geq 0 \\ x_2 &\geq 0. \end{aligned} \tag{1}$$

The left hand side of the first inequality represents the amount of iron the person consumes when eating x_1 units of cereal and x_2 units of steak; it must be at least 60 as otherwise not enough iron is consumed. Similarly the left hand side of the second inequality represents the amount of protein consumed, and must be at least 70. The last two inequalities represent the fact that we cannot eat a negative amount of a food. The cost of the diet is

$$\text{Cost}(x_1, x_2) \ = \ 20x_1 + 2x_2, \tag{2}$$

and the diet problem becomes: *Minimize* $\mathrm{Cost}(x_1, x_2)$*, subject to* x_1, x_2 *satisfy* (1). *It is very important that, not only are the constraints linear in the variables, but the function we are trying to maximize is linear as well.*

We may rewrite (1) in matrix form. Let

$$A = \begin{pmatrix} 30 & 5 \\ 15 & 10 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad b = \begin{pmatrix} 60 \\ 70 \end{pmatrix}. \tag{3}$$

Then the diet problem is equivalent to

$$\text{Minimize } 20x_1 + 2x_2, \text{ subject to } Ax \geq b \text{ and } x \geq 0. \tag{4}$$

The above is an example of a Linear Programming problem:

1. we have variables $x_j \geq 0$ for $j \in \{1, \dots, N\}$;
2. the variables satisfy linear constraints, which we can write as $Ax \geq b$;
3. the goal is to minimize a *linear* function of the variables: $c^T x = c_1x_1 + \cdots + c_Nx_N$.

Note the similarity between (4) and a standard linear algebra problem. The differences are that, instead of $Ax = b$ we have $Ax \geq b$, and instead of solving for x with $Ax = b$ we are solving for x satisfying $Ax \geq b$ which minimizes some linear function. Thus Linear Algebra becomes a subset of Linear Programming. In fact, in the next section we show how, by introducing additional variables, we may replace the constraints $Ax \geq b$ with new constraints and new variables, $A'x' = b'$.

2.2.2 Definitions

Before defining the canonical Linear Programming problem, we first observe that it suffices to consider *only* cases where the constraints are greater than or equal to. This is because a constraint such as

$$a_{i1}x_1 + \cdots + a_{iN}x_N \leq b_i. \tag{5}$$

may be re-written as

$$-a_{i1}x_1 - \cdots - a_{iN}x_N \geq -b_i; \tag{6}$$

the entries of the matrix A are allowed to be any real number. Thus there is no loss in generality in assuming all the constraints are greater than or equal to. Further, at the cost of adding additional variables, we may assume all the constraints are actually equalities.

Consider some constraint

$$a_{i1}x_1 + \cdots + a_{iN}x_N \geq b_i. \tag{7}$$

We introduce a new variable $z_i \geq 0$, and change the above constraint to

$$-a_{i1}x_1 - \cdots - a_{iN}x_N + z_i = b_i. \tag{8}$$

Thus for each constraint in $Ax \geq b$ with a greater than sign, at the cost of adding a new variable $z_i \geq 0$ we may replace the greater than sign with an equality. The variable z_i is added *only* to a constraint, not to the linear function we are trying to minimize.

We note that we may assume each variable $x_j \geq 0$. To do this may require adding additional constraints and additional variables (and the additional variables will also be non-negative). Assume we want $x_j \geq m_j$ (we allow m_j to equal $-\infty$). If $m_j \geq 0$ and is finite then we simply add the constraint $x_j \geq m_j$. If $m_j \leq 0$ and is finite we replace the variable x_j with z_j by setting $z_j = x_j - m_j$ with $z_j \geq 0$; note we still have a linear function to minimize. Finally, if $m_j = -\infty$ we introduce two new variables $u_j, v_j \geq 0$, and we replace x_j with $u_j - v_j$.

Finally, say that instead of minimizing the linear function $c^T x$ we want to maximize it. As minimizing the linear function $-c^T x$ is the same as maximizing the function $c^T x$, there is no loss in generality in assuming we want to minimize a linear function.

The above arguments shows that we may take *any* Linear Programming problem and write it in the following form:

Definition 2.1 (Canonical Linear Programming Problem)**.** *The canonical Linear Programming problem is of the following form:*

1. *we have variables $x_j \geq 0$ for $j \in \{1, \dots, N\}$;*
2. *the variables satisfy linear constraints, which we can write as $Ax = b$ (where A is a matrix with M rows and N columns, and b is a column vector with M components);*
3. *the goal is to minimize a linear function of the variables: $c^T x = c_1 x_1 + \cdots + c_N x_N$.*

If x satisfies the constraints ($Ax = b$, $x \geq 0$) then we call x a ***feasible*** solution to the canonical Linear Programming problem; if further x minimizes the linear function $c^T x$, then x is called an ***optimal*** solution to the canonical Linear Programming problem.

We discuss some pathological cases. Consider the following canonical Linear Programming problems.

1. The constraints are $x_1 = -2007$, with $x_1 \geq 0$ and minimize $10x_1$. There are no feasible solutions; thus there are no optimal solutions.
2. The constraints are $2x_1 - 5x_2 = 0$, with $x_1, x_2 \geq 0$ and minimize $-17x_1$. There are infinitely many feasible solutions: any (x_1, x_2) works with $x_1 = 2.5x_2$; however, there is no optimal solution (send $x_1 \to \infty$).
3. The constraints are $x_1 + x_2 = 1$, with $x_1, x_2 \geq 0$ and minimize $x_1 + x_2$. Here there are infinitely many feasible solutions, and each feasible solution is also an optimal solution.

The above examples show some care is required. A general Linear Programming problem need not have a feasible solution. If it does have a feasible solution, it need not have an optimal solution. Further, even if it does have an optimal solution, it need not have a unique optimal solution.

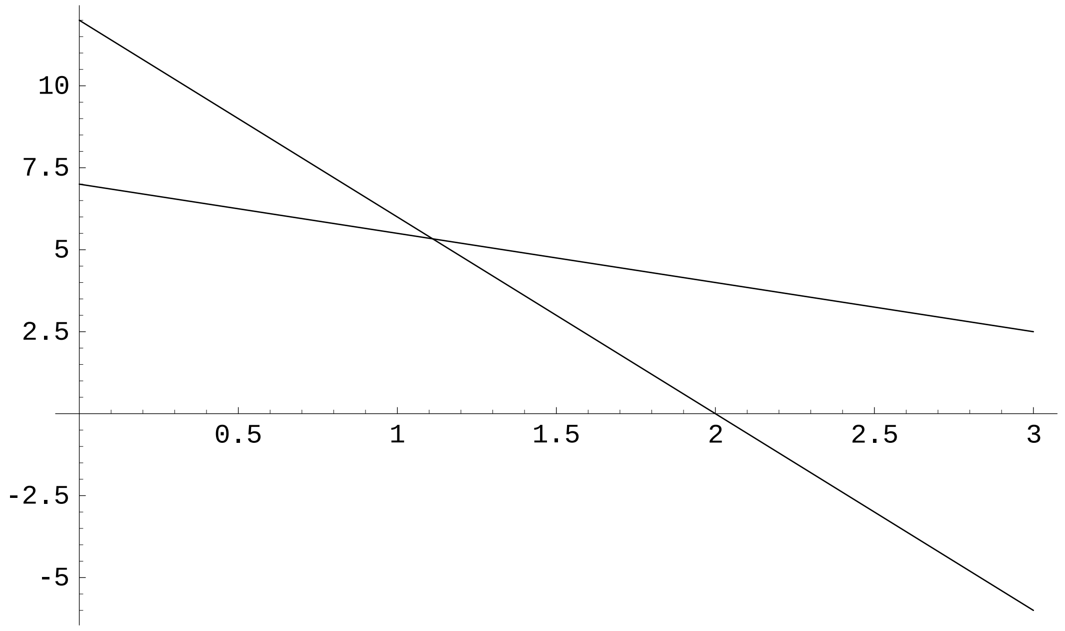

Figure 1: Plot of constraints for Diet Problem

Exercise 2.2. *Assume there are P plants (i.e., Alaska, Texas, Mexico, ...) that produce oil and there are M markets that need oil (i.e., Boston, London, Tokyo, ...). Let c_{ij} denote the cost of shipping one barrel of oil from plant i to market j (thus c_{12} is the cost of shipping one barrel of oil from our plant in Texas to London). Assume city j needs d_j barrels of oil each day, and plant i can supply s_i barrels of oil a day. Write the Linear Programming problem corresponding to this situation: find the constraints, and find the quantity to be minimized.*

2.2.3 Solution to the Diet Problem

In Figure 1 we graph the four constraints to the Diet Problem considered earlier (see (4); A, x and b are defined in (3)). Before finding an optimal solution, we first find all possible feasible solutions. Note that if (x_1, x_2) is a feasible solution than it must be in the region in the first quadrant above both lines. Thus there are infinitely many candidates for the optimal solution (or solutions!).

Fortunately, we may greatly winnow down the candidate list for optimal solutions. The idea is somewhat similar in spirit to optimization problem from calculus; here we first show the optimal solutions *cannot* occur in the interior and therefore *must* occur on the boundary of the polygon, and then show that they *must* occur at a vertex.

We are trying to minimize the linear cost function $20x_1 + 2x_2$. Consider the function $\text{Cost}(x_1, x_2) = 20x_1+2x_2$. We look at contours where the function is constant: $\text{Cost}(x_1, x_2) = c$; we sketch some constant cost contours in Figure 2. Note that the contours of constant cost are parallel; this is clear as they are of the form $20x_1 + 2x_2 = c$. Further, the smaller the value of c,

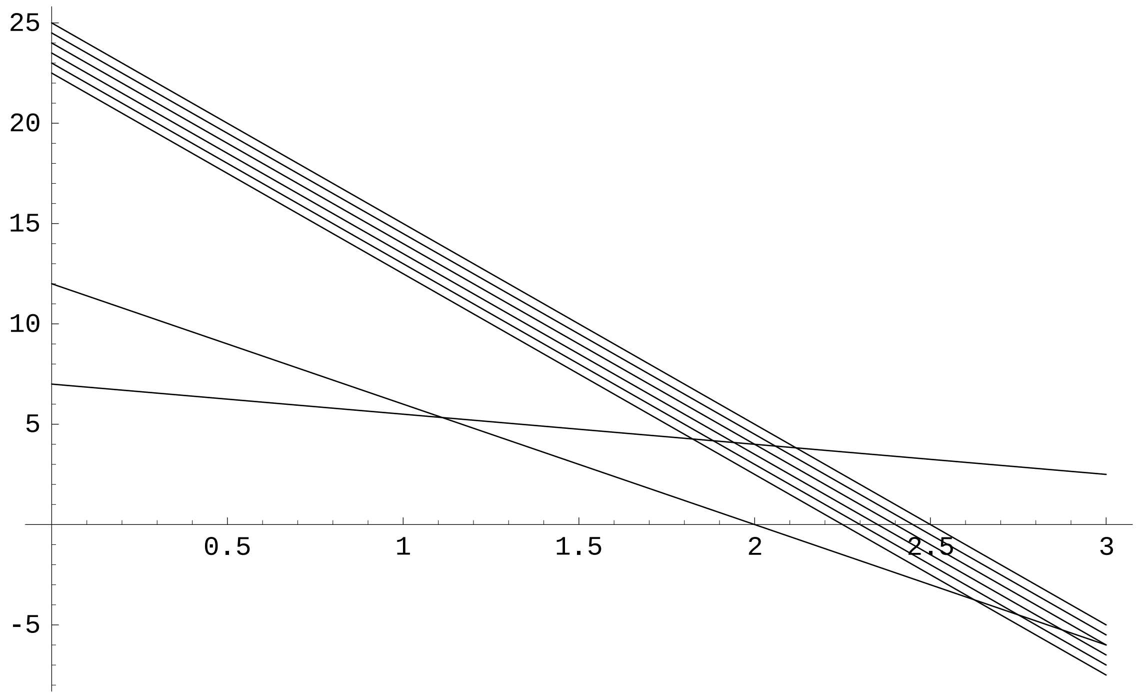

Figure 2: Plot of constraints for Diet Problem

the smaller the cost. Thus, given the choice between two lines, to minimize cost we choose the *lower* line.

Therefore, if we start at any point *inside* the polygon (the set of feasible solutions), by flowing on a line of constant cost we may move to a point on the boundary with the same cost. Thus, to find *an* optimal solution, it suffices to check the feasible solutions on the boundary; this is a general feature of linear programming problems.

Additionally, it is enough to check the *vertices* of the polygon. This is because if we are on one of the edges of the polygon, if we move to the left the cost decreases. Thus it suffices to check the three vertices to find the cheapest diet which contains the minimum daily requirements. Because of the cost of a unit of cereal ($20) and steak ($2), the slope of the line is such that the cheapest diet has *no* cereal, and only steak. This is why we chose such unreasonable numbers for the cost of cereal and steak, so that we could check our first solution with our intuition.

Let us now consider a more reasonable set of prices. Let the cost function be $\mathrm{Cost}(x_1, x_2) = x_1 + x_2$ (so the two products both cost $\$1$ per unit). We plot some cost lines in Figure 3. Note now that the optimal solution, while still one of the vertex points, is *not* $x_1 = 0$.

Exercise 2.3. *Find the optimal solution to the Diet Problem when the cost function is* $\mathrm{Cost}(x_1, x_2) = x_1 + x_2$.

Exercise 2.4. *There are three vertices on the boundary of the polygon (of feasible solutions); we have seen two choices of cost functions that lead to two of the three points being optimal*

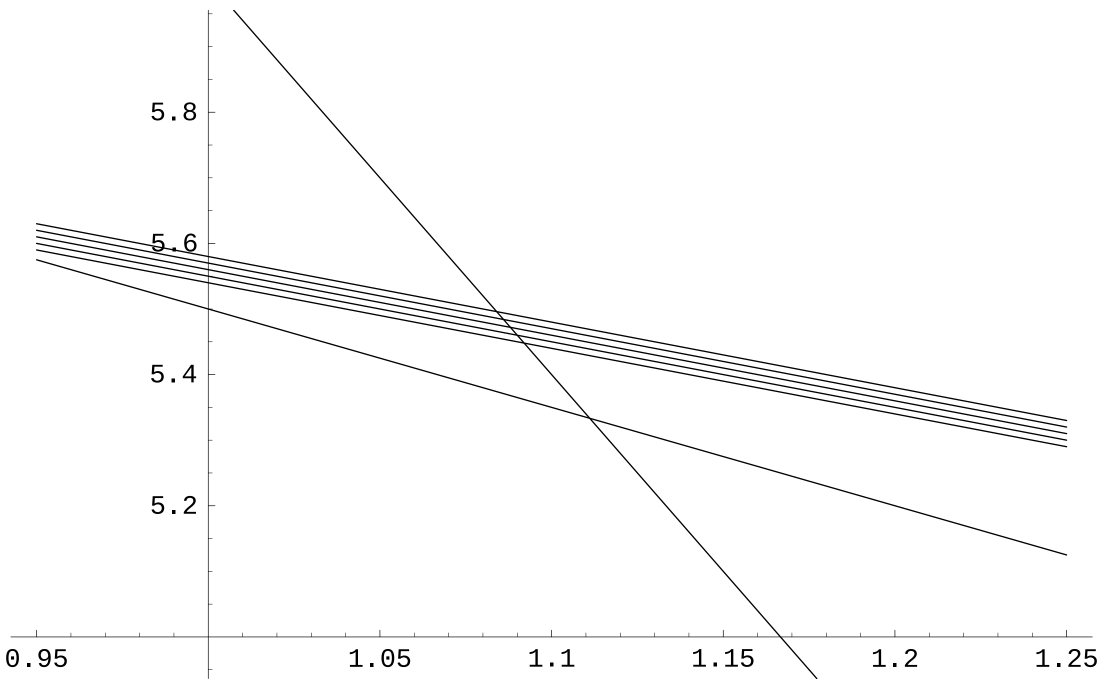

Figure 3: Plot of constraints for Diet Problem

solutions; find a linear cost function which has the third vertex as an optimal solution.

Exercise 2.5. *Generalize the diet problem to the case when there are three or four types of food, and each food contains one of three items a person needs daily to live (for example, calcium, iron, and protein). The region of feasible solutions will now be a subset of $\mathbb{R}^3$. Show that an optimal solution is again a point on the boundary.*

2.3 Duality and Basic Solutions

2.3.1 Duality

Recall the canonical Linear Programming problem may be assumed to be in the following form:

1. variables $x = (x_1, \dots, x_N) \geq 0$;
2. linear constraints $Ax = b$, with $b = (b_1, \dots, b_M)$;
3. minimize the linear function $c^T x$.

Definition 2.6 (Dual Problem)**.** *Given a canonical Linear Programming problem, the Dual Problem is defined by*

1. *variables $y = (y_1, \dots, y_M) \in \mathbb{R}^M$;*
2. *linear constraints $y^T A \leq c^T$;*

3. maximize the linear function $y^T b$.

We shall see later that it is often useful to pass from the original problem to the dual problem; we give an example here to show how a dual problem can often be easier to study. Consider the following chess problem: place 5 queens on a 5×5 chessboard such that there are three squares where we may place pawns so that no queen attacks any pawn (remember a queen attacks horizontally, vertically, and diagonally). We depict one configuration of five queens which allows three pawns to be placed safely on a 5×5 board:

P			P	
P				
				Q
	Q			Q
	Q	Q		

(9)

One way to attack the problem is to look at all the different ways 5 queens may be placed on the 5×5 chessboard. Sadly, however, there are $\binom{25}{5} = 53,130$ possibilities to check! One can cut down on this by exploiting symmetry (there are only 6 inequivalent places to place the first queen, not 25), though even if we do this, the number of possibilities is large enough so that it is undesirable to check all by hand.

The principle of duality is applicable here – we replace the problem we are studying with an easier, equivalent one. Instead of trying to put down 5 queens so that 3 pawns can be placed safely on the 5×5 chessboard, consider the *Dual Problem*, where we try to place 3 queens on the board so that 5 pawns can be placed safely. Why is this equivalent? If we are able to do this, say the 3 queens are at (x_i, y_i) (for $i \in \{1, 2, 3\}$) and the 5 pawns are at (u_i, v_i) (for $i \in \{1, \dots, 5\}$). Then all we need do is replace each pawn at (x_i, y_i) with a queen, and each queen at (u_i, v_i) with a pawn.

The advantage is clear: rather than having to investigate $\binom{25}{5}$ possibilities, now we need only study $\binom{25}{3} = 2,300$ (and we may further reduce the number of possibilities by symmetry arguments).

Exercise 2.7. *Prove the Dual Problem of the Dual Problem is the original Linear Programming problem.*

We give a simple example of how the Dual Problem can provide information about the original problem. Assume y is a feasible solution of the Dual Problem. Thus $y^T A \leq c^T$. We immediately find that, if x is a feasible solution of the original Linear Programming problem, then

$$y^T A x \;=\; (y^T A) x \;\leq\; c^T x \tag{10}$$

and

$$y^T A x \;=\; y^T (A x) \;=\; y^T b. \tag{11}$$

Thus

$$y^T b \;\leq\; c^T x, \tag{12}$$

and any feasible solution of the Dual Problem gives a lower bound for the linear function we are trying to maximize in the canonical Linear Programming problem. This leads to the following test for optimality:

Lemma 2.8. *Consider a canonical Linear Programming problem with a feasible solution $\widehat{x}$, and its Dual Problem with a feasible solution $\widehat{y}$. If $c^T\widehat{x} = \widehat{y}^T b$ then $\widehat{x}$ is also an optimal solution.*

Proof. Let x be any feasible solution to our Linear Programming problem. From (12) we know that $\widehat{y}^T b \leq c^T x$; however, by assumption we have $c^T\widehat{x} = \widehat{y}^T b$, which implies that $c^T\widehat{x} \leq c^T x$ for x *any* feasible solution to our Linear Programming problem. Thus $\widehat{x}$ is an optimal solution, minimizing our linear function. □

2.3.2 Basic Solutions

Let x be a feasible solution for the canonical Linear Programming problem with constraints $Ax = b$. While all the coordinates of x are non-negative, some may be zero. Let x_{j_1}, x_{j_2}, $\dots, x_{j_k}$ denote the coordinates of x that are *positive*. If the corresponding columns $A_{j_1}, \dots, A_{j_k}$ are linearly independent, then we say that x is a **basic solution**[1] to the Linear Programming problem. The following simple observation is crucial to efficiently solving Linear Programming problems.

Lemma 2.9. *There are only finitely many basic solutions to a canonical Linear Programming problem.*

Proof. Let A have M rows and N columns. There are only finitely many subsets of columns (in fact, there are $2^N - 1$ non-empty subsets of k columns of A, $1 \leq k \leq N$). Consider one such subset; let $A_{j_1}, \dots, A_{j_k}$ denote a set of k linearly independent columns of A. We are reduced to showing that there are only finitely many $(x_{j_1}, \dots, x_{j_k})$ (with each coordinate positive) such that

$$A_{j_1}x_{j_1} + \cdots + A_{j_k}x_{j_k} \ = \ b. \tag{13}$$

We do this by showing there is at most one such solution. We may rewrite (13) as

$$A'x' \ = \ b, \tag{14}$$

where $A' = (A_{j_1}\ A_{j_2}\ \cdots\ A_{j_k})$ has M rows and k columns. Note $M \geq k$, because if $M < k$ then the k columns cannot be linearly independent. We would like to say $x' = A'^{-1}b$; however, A' is not necessarily a square matrix (and if it is not a square matrix, it cannot be invertible). We remedy this by multiplying both sides by A'^T, obtaining

$$A'^T A'x' \ = \ A'^T b; \tag{15}$$

as $M \geq k$, the $k \times k$ matrix $A'^T A'$ is invertible. Therefore

$$x' \ = \ (A'^T A')^{-1} A'^T b, \tag{16}$$

proving that for every set of k linearly independent columns, there is at most one basic solution (there is only a basic solution if all the coordinates of x' are positive). □

[1]We assume that $b \neq 0$; if $b = 0$ then $x = 0$ is also considered a basic solution.

For example, if

$$A \;=\; \begin{pmatrix} 1 & 2 & 3 \\ 2 & 5 & 7 \\ 2 & 3 & 5 \end{pmatrix}, \tag{17}$$

then we may take A' to be the first two columns, and we find

$$A'^T A' \;=\; \begin{pmatrix} 9 & 18 \\ 18 & 38 \end{pmatrix}, \quad \det(A'^T A') \;=\; 18. \tag{18}$$

Exercise 2.10. *Prove that if A' has M rows and k columns, with $M \geq k$, then $A'^T A'$ is invertible.*

The above arguments show that there are only finitely many basic solutions to a canonical Linear Programming problem. We will show that whenever there is a feasible solution then there is a basic feasible solution (i.e., a feasible solutions which is also a basic solution); similarly, we will show that whenever there is an optimal solution then there is also a basic optimal solution. Thus, rather than having to check infinitely many possibilities, we are reduced to checking a finite set.

While this is a terrific simplification (any finite number is significantly less than infinity!), in order for this to be *useful* we must have an efficient algorithm for checking all the basic feasible and basic optimal solutions. In many problems M and N (the size of the constraints matrix) are quite large. The number of possible basic solutions (remember that if a basic solution has k components then $M \geq k$) can be as large as $\sum_{k=1}^{M} \binom{N}{k}$. For large M and N, it is infeasible to directly check each possibility.

Exercise 2.11. *For fixed M, find some lower bounds for the size of $\sum_{k=1}^{M} \binom{N}{k}$. If $M = N = 1000$ (which can easily happen for real world problems), how many basic feasible solutions could there be? There are less than 10^{90} sub-atomic objects in the universal (quarks, photons, et cetera). Assume each such object is a supercomputer capable of checking 10^{20} basic solutions a second (this is much faster than current technology!). How many years would be required to check all the basic solutions?*

Theorem 2.12. *Consider a canonical Linear Programming problem. If there is a feasible solution, then there is a basic feasible solution; if there is an optimal solution, then there is a basic optimal solution.*

Proof. Choose *any* feasible solution with the fewest number of positive components (remember each component is non-negative). By assumption a feasible solution exists; the number of positive components is between 0 and N, and thus it makes sense to talk about feasible solutions with as few positive components as possible. For definiteness, say that the fewest positive components a feasible solution has is k. Let x be a feasible solution with k positive components, say $x_{j_1}, \dots, x_{j_k}$. We must show that the corresponding columns $A_{j_l}, \dots, A_{j_k}$ of A are linearly independent. Assume not. Then there are numbers γ_{j_ℓ} such that

$$\gamma_{j_1} A_{j_1} + \cdots + \gamma_{j_k} A_{j_k} \;=\; 0; \tag{19}$$

without loss of generality we may assume γ_{j_1} is non-zero and positive (by relabeling we may assume $\gamma_{j_1} \neq 0$, as at least two of the γ_{j_ℓ}'s are non-zero; by possibly multiplying by -1 we may ensure that $\gamma_{j_1} > 0$). Since x is a feasible solution, $Ax = b$. As the components of x that are zero do not matter, $Ax = b$ is the same as

$$x_{j_1}A_{j_1} + \cdots + x_{j_k}A_{j_k} \ = \ b. \tag{20}$$

We multiply (19) by λ and subtract this from the previous equation, obtaining

$$(x_{j_1} - \lambda\gamma_{j_1})A_{j_1} + \cdots + (x_{j_k} - \lambda\gamma_{j_k})A_{j_k} \ = \ b. \tag{21}$$

If $\lambda = 0$ then all the $x_{j_\ell} - \lambda\gamma_{j_\ell}$ are positive, hence by continuity these will still be positive if $|\lambda|$ is small. We take the largest λ such that (21) holds with all components non-negative. Such a λ exists, as $\lambda \leq x_{j_1}/\gamma_{j_1}$ (which is positive as x_{j_1} and γ_{j_1} are positive). The largest such λ results in one of the $x_{j_\ell} - \lambda\gamma_{j_\ell}$ equaling zero. Thus we obtain a feasible solution to the Linear Programming problem with at most $k-1$ positive components (there might be fewer positive components if two or more $x_{j_\ell} - \lambda\gamma_{j_\ell}$ vanish for the maximal λ); this contradicts the minimality of x. Therefore the columns $A_{j_1}, \dots, A_{j_k}$ are linearly independent, and the existence of a feasible solution implies the existence of a basic feasible solution.

The proof of the existence of an optimal solution implying the existence of an optimal solution proceeds similarly. Let x be an optimal solution with fewest positive components (say $x_{j_1}, \dots, x_{j_k}$). The proof is completed by showing the corresponding columns $A_{j_1}, \dots, A_{j_k}$ are linearly independent. As before, we may find λ and γ_{j_ℓ}'s such that

$$(x_{j_1} - \lambda\gamma_{j_1})A_{j_1} + \cdots + (x_{j_k} - \lambda\gamma_{j_k})A_{j_k} \ = \ b; \tag{22}$$

we may assume $\gamma_{j_1} > 0$. Thus (in obvious notation) $x - \lambda\gamma$ is a new feasible solution, with cost

$$c^T x - \lambda(c_{j_1}\gamma_{j_1} + \cdots + c_{j_k}\gamma_{j_k}); \tag{23}$$

for $|\lambda|$ sufficiently small all components of $x - \lambda\gamma$ are positive. The new cost must equal the old cost, as x is an optimal solution (if the new cost did not equal the old cost, by choosing the sign of λ appropriately, for small λ we could produce a new feasible solution with lower cost, contradicting x being an optimal solution). Thus

$$c_{j_1}\gamma_{j_1} + \cdots + c_{j_k}\gamma_{j_k} \ = \ 0. \tag{24}$$

As before, we take the largest λ such that $x - \lambda\gamma$ has non-negative components (note $\lambda \leq x_{j_1}/\gamma_{j_1}$). The cost of all these solutions are independent of λ, and equal to the minimum cost. Thus we have found a new optimal solution with at most $k-1$ positive components, contradicting the minimality of x; thus the columns $A_{j_1}, \dots, A_{j_k}$ are linearly independent and there is a basic optimal solution. $\square$

Note the above proof is non-constructive proof by contradiction[2]: there are basic feasible

[2]For another example of a non-constructive proof by contradiction, recall Euclid's proof of the infinitude of primes. Assume there are only finitely many primes, say $p_1, \dots, p_N$. Consider the number $p_1 \cdots p_N + 1$. If this is prime, our list is incomplete and we are done. If it is composite, it must be divisible by a prime; however, as it has remainder 1 upon division by $p_1, p_2, \dots, p_N$, it must be divisible by a prime not in our list, and again we obtain that our list is incomplete. Therefore there are infinitely many primes, though we have not constructed infinitely many primes.

and basic optimal solutions, but we do not know what they are. Later we shall see how to (efficiently!) find them.

2.4 Solving the Canonical Linear Programming Problem: The Simplex Method

Our arguments above show that it suffices to check finitely many potential solutions to find the optimal solution (if one exists!) of a canonical Linear Programming problem. We now describe the Simplex Method, which is an efficient way to find optimal solutions. See [Da, Fr] for more details.

We assume we are studying a non-degenerate canonical Linear Programming problem, and we make the following assumptions from here on:

- If A has M rows and N columns, then $M < N$. This implies that there are more unknowns then equations, so the system $Ax = b$ is undetermined and can have infinitely many solutions; if $M \leq N$ there is at most one solution.
- The M rows of A are linearly independent. If the rows are not linearly independent, then either we cannot solve $Ax = b$, or if we can then at least one of the rows is unnecessary.
- We assume b is not a linear combination of fewer than M columns of A. If b is a combination of fewer than M columns, this will create a technical difficulty in the simplex method. Fortunately this is a very weak condition: if we change some of the entries of b by small amounts (less than 10^{-10}, for example[3]), this should suffice to break the degeneracy.

The simplex method has two phases:

1. Phase I: Find a basic feasible solution (or prove that none exists);
2. Phase II: Given a basic feasible solution, find a basic optimal solution (or prove none exists). If no optimal solution exists, Pase II produces a sequence of feasible solutions with cost tending to minus infinity.

We now describe the algorithms for these steps, and then comment on the run-time. Remember it is not enough to find an optimal solution – we need to find an optimal solution in a reasonable amount of time! However, for many complicated systems, instead of finding optimal solutions we must lower our goals and find an almost optimal solution.

For example, imagine we own a major airline. Contrary to what many people might believe, our goal is *not* to fly people from one city to another; our goal is to maximize profit (which is done by flying people, of course). Thus we would be willing to fly less people if we can charge more. We have many planes, and can estimate the demand for flying between cities and what people will pay for flights. We have numerous constraints, ranging from the obvious (once a

[3]There might be some problems with changes this small, as computers can have round-off errors and we may need to worry about numerical precision. In practice, however, this condition almost always met, and we shall assume it holds.

plane takes off, we cannot use it for another flight until it lands), to ones imposed on us (a flight crew must rest for a minimum of H hours before they are allowed to fly another plane). We need to find a schedule each day; further, we must announce these schedules far enough in advance so that people may purchase tickets. Let us assume we know that the maximum possible profit we can make in a month is \$150,000,000. After a few hours of computation we find a schedule that would earn us \$149,999,982, and we calculate that it would take a year to check all the remaining possible schedules. While this is not an optimal schedule, it is so close (differing by just \$18), that it is not worth waiting a year (and thus flying no planes and earning no revenue) to potentially earn at most another \$18 for that month.

2.4.1 Phase I of the Simplex Method

We first show how, if we can do Phase II of the simplex method, then we can do Phase I; in the next section we will show how to do Phase 2.

Thus our goal is to find a basic feasible solution (or show none exists) to the canonical Linear Programming problem

$$Ax \ = \ b, \quad x \geq 0, \quad \min_x c^T x. \tag{25}$$

We assume we know how to do Phase II, namely given a basic feasible solution to a canonical Linear Programming problem, we can find a basic optimal one (or prove none exists by constructing a sequence of feasible solutions with costs tending to minus infinity).

The constraints of the canonical Linear Programming problem are

$$\sum_{j=1}^{N} a_{ij} x_j \ = \ b_i, \quad i \in \{1, \dots, M\}, \tag{26}$$

where the variables x_j are non-negative. Consider now the following canonical Linear Programming problem (which is clearly related to our initial problem):

1. the variables are $x_j \geq 0$ for $j \in \{1, \dots, N\}$ and $z_i \geq 0$ for $i \in \{1, \dots, M\}$;

2. the linear constraints are

$$\left(\sum_{j=1}^{N} a_{ij} x_j\right) + z_i \ = \ b_i, \quad i \in \{1, \dots, M\}, \tag{27}$$

   which can be written as $A'x' = b$ with $A' = (A\ I)$ a matrix with first N columns those of A and final M columns the $M \times M$ identity matrix, and $x' = (x_1, \dots, x_N, z_1, \dots, z_M)$.

3. the linear function to minimize is

$$z_1 + \cdots + z_m. \tag{28}$$

Remember the goal is to find a basic feasible solution to (25), and we are assuming we know how to do Phase II (given a basic feasible solution, we can find a basic optimal solution, or prove one does not exist by constructing a sequence of feasible solutions with costs tending to minus infinity). For the new canonical Linear Programming problem, it is easy to find a basic feasible solution: take $x_j = 0$ for each j, and[4] $z_i = b_i$ for each i. This is clearly a feasible solution; to show that it is a basic solution we must show that the columns of the matrix $A' = (A\ I)$ corresponding to the positive components of the feasible solution are linearly independent. This follows immediately from the fact that the only non-zero components are among the z_i, and the corresponding columns are columns of an $M \times M$ identity matrix and therefore linear independent.

We now perform Phase II to the basic feasible solution

$$(x_1, \dots, x_N, z_1, \dots, z_M) \;=\; (0, \dots, 0, b_1, \dots, b_M). \tag{29}$$

This problem *must*[5] have an optimal solution, which we denote

$$(x_{\text{op},1}, \dots, x_{\text{op},N}, z_{\text{op},1}, \dots, z_{\text{op},M}). \tag{30}$$

There are two cases for our optimal solution:

1. If
$$\min_z (z_1 + \cdots + z_M) \;=\; z_{\text{op},1} + \cdots + z_{\text{op},M} \;=\; 0, \tag{31}$$
then we have found a basic feasible solution to the original Linear Programming problem, namely the first N components of (30);

2. If
$$\min_z (z_1 + \cdots + z_M) \;=\; z_{\text{op},1} + \cdots + z_{\text{op},M} \;>\; 0 \tag{32}$$
(it clearly cannot be negative as each $z_i \geq 0$), there is no basic feasible solution to the original Linear Programming problem (if there were a basic feasible solution to the original Linear Programming problem, that would lead to a lower cost for the related problem, as that would allow us to take each $z_i = 0$ and thus reduce the cost to zero).

We are therefore reduced to showing how to do Phase II.

2.4.2 Phase II of the Simplex Method

We now describe an algorithm for Phase II, namely how to pass from a basic feasible solution to a basic optimal solution (or prove one does not exist by constructing a sequence of basic feasible solutions with costs tending to minus infinity). We first note

Lemma 2.13. *Let x be a basic feasible solution; x must have exactly M non-zero entries.*

[4]We must be careful, as each z_i must be non-negative. Without loss of generality we may assume each b_i in the initial Linear Programming problem is non-negative, as if any were negative we could multiply the corresponding constraint by -1.

[5]The cost is just $z_1 + \cdots + z_M$, and $z_i \geq 0$. Thus the cost is non-negative. If there were no optimal solution, then Phase II would produce a sequence of solutions with cost tending to minus infinity; as the cost cannot be less than zero, this cannot happen. Therefore there is an optimal solution for this Linear Programming problem.

Proof. This follows from our assumptions (A is an $M \times N$ matrix with $M \leq N$, b is a vector with M components which cannot be written as a linear combination of fewer than M columns of A).

We first show that x has at most M positive entries. The rank of A is at most M, and the columns corresponding to the positive entries of the feasible solution x must be independent. Thus x cannot have more than M positive components.

Further, x must have at least M positive components; if it did not, then b could be written as the sum of fewer than M columns of A. $\square$

We continue with the description of how to perform Phase II. We may assume that we have a basic feasible solution x with exactly M non-zero entries. Let $B = \{j : x_j > 0\}$; note $|B| = M$ by the above lemma. We call B the basis. Let $x_B = (x_{j_1}, \dots, x_{j_m})$ be the positive components of x, and let A_B denote the matrix of columns A_j of A where $j \in B$; we call A_B the basis matrix. Thus we have

$$A_B x_B \ = \ b. \tag{33}$$

Further, A_B is an *invertible* matrix (it is an $M \times M$ matrix with M linearly independent columns). Thus we can also study the linear system of equations

$$y^T A_B \ = \ c_B^T, \tag{34}$$

which has the unique solution

$$y \ = \ c_B^T A_B^{-1}. \tag{35}$$

There are two possibilities for this vector y: either it is a feasible solution to the Dual Problem (the dual of the original Linear Programming problem), or it is not a feasible solution to the Dual Problem.

Case 1: y is feasible for the Dual Problem. (See Definition 2.6 for the statement of the Dual Problem.) If y is feasible[6] for the Dual Problem, then from (33) and (34) we have

$$y^T b \ = \ y^T A_B x_B \ = \ c_B^T x_B \ = \ c^T x. \tag{36}$$

By Lemma 2.8, the fact that $y^T b = c^T x$ means that x is an optimal solution of our Linear Programming problem. As x is a basic feasible solution, this means x is a basic optimal solution.

Case 2: y is not feasible for the Dual Problem. As y is not a feasible solution for the Dual Problem, for some s we have $y^T A_s > c_s$; by construction we know $s \notin B$. The idea is that we can lower the cost by bringing the column A_s into the basis matrix A_B.

As the M columns of the $M \times M$ matrix A_B are linearly independent, and A_s is a vector with M components, we have

$$A_s \ = \ \sum_{j \in B} t_j A_j, \tag{37}$$

or

$$A_s \ = \ A_B t, \quad t \ = \ A_B^{-1} A_s. \tag{38}$$

[6]It is easy to check and see if y is feasible. We need only check that $y^T A_j \leq c_j$ for all $j \in \{1, \dots, M\}$. By construction this holds for $j \in B$; thus we need only check these conditions for $j \notin B$.

From the relations

$$\sum_{j\in B} x_j A_j \;=\; b, \quad A_s - \sum_{j\in B} t_j A_j \;=\; 0, \tag{39}$$

we find that

$$\lambda A_s + \sum_{j\in B} (x_j - \lambda t_j) A_j \;=\; b; \tag{40}$$

for λ sufficiently small and positive, we have a new feasible solution x' to the original Linear Programming problem, with

$$x'_j \;=\; \begin{cases} \lambda & \text{if } j = s \\ x_j - \lambda t_j & \text{if } j \in B \\ 0 & \text{otherwise.} \end{cases} \tag{41}$$

The original cost (associated to the feasible solution x) is

$$\sum_{j\in B} x_j c_j; \tag{42}$$

the new cost (associated to the feasible solution x') is

$$\lambda c_s + \sum_{j\in B} (x_j - \lambda t_j) c_j. \tag{43}$$

We now show that the new cost is less than the old cost. The new cost minus the old cost is

$$\lambda \left(c_s - \sum_{j\in B} t_j c_j \right); \tag{44}$$

as $\lambda > 0$, we need only show that $c_s < \sum_{j\in B} t_j c_j$ to show the new cost is less than the old cost. From (34) we have $y^T A_B = c_B^T$, and from (38) we have $A_s = A_B t$. These relations imply

$$\sum_{j\in B} t_j c_j \;=\; y^T A_B t \;=\; y^T A_s \;>\; c_s, \tag{45}$$

where the last inequality follows from our assumption that y is not feasible for the Dual Problem[7].

There are two possibilities: either all $t_j \leq 0$, or at least one is positive.

1. **Case 2: Subcase (i):** Assume all $t_j \leq 0$ in (38). Then we may take λ to be *any* positive number in (40), and each positive λ gives us another feasible solution. Therefore the cost in (44) tends to minus infinity as λ tends to infinity. This implies we can construct a sequence of feasible solutions to the original Linear Programming problem with costs tending to minus infinity, and therefore the original Linear Programming problem does not have an optimal solution.

[7]The assumption that y is not feasible for the Dual Problem means that $y^T A_s > c_s$.

2. **Case 2: Subcase (ii):** Suppose now at least one $t_j > 0$. The largest positive λ we may take and still have a feasible solution is

$$\lambda^* \;=\; \min_{j \in B} \left(\frac{x_j}{t_j} : t_j > 0 \right). \tag{46}$$

We may assume the minimum for λ^* occurs for $j = p$. Note that p is unique; if not, we would find that b is a linear combination of fewer than M columns, contradicting our assumption on b (we are basically swapping the p^{th} column for the s^{th} column). We have thus found a new feasible basic solution (we leave it to the reader to check that our solution, in addition to being feasible, is also basic) with exactly M non-zero entries. We now restart Phase II with our new basic feasible solution as our basic feasible solution, and continue the search for a basic optimal solution.

We have described the algorithm for Phase II of the simplex method; we show that it must terminate either in an optimal solution, or by constructing a sequence of basic feasible solutions with costs tending to minus infinity. When we start Phase II, three things can happen: (1) we end up in Case 1: if this happens, then we have found an optimal solution; (2) we end up in Case 2, Subcase (i): if this happens, there is no optimal solution (and we have a sequence of basic feasible solutions with costs tending to minus infinity); (3) we end up in Case 2, Subcase (ii): if this happens, we restart Phase II with the new basic feasible solution.

The only way Phase II would not terminate is if we always end up in Case 2, Subcase (ii) each time we apply it. Fortunately, we can see this cannot occur. A is an $M \times N$ matrix ($M < N$). There are only finitely many sets of M linearly independent columns of A (there are at most $\binom{N}{M}$). Each time we enter Case 2, Subcase (ii) we obtain a new feasible solution with cost *strictly* less than the previous cost. This implies, in particular, that all the solutions from Case 2, Subcase (ii) are distinct. As there are only finitely many possibilities, eventually Phase II must terminate with either a basic optimal solution to the original Linear Programming problem, or with a sequence of feasible solutions with costs tending to minus infinity. This completes our analysis of the simplex method.

Exercise 2.14. *Consider the following Linear Programming problem: $x_j \geq 0$,*

$$\begin{pmatrix} 1 & 4 & 5 & 8 & 1 \\ 2 & 2 & 3 & 8 & 0 \\ 3 & 2 & 1 & 6 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 311 \\ 389 \\ 989 \end{pmatrix}, \tag{47}$$

and we want to minimize

$$5x_1 + 8x_2 + 9x_3 + 2x_4 + 11x_5. \tag{48}$$

Find (or prove one does not exist) an optimal solution.

2.4.3 Summary of the Simplex Method

While we have shown that the simplex method will terminate (with either a basic optimal solution or a sequence of feasible solutions with cost tending to minus infinity), we have *not* shown that it will terminate in a reasonable amount of time! It is imperative we show this if we are interested in using the simplex method as a practical tool, and not merely to note it as a theoretical curiosity. We encourage the interested reader to peruse [Fr] and the references therein for more details, especially proofs of the efficiency of these methods.

2.5 Binary Integer Linear Programming

We have merely scratched the surface of a rich and very useful theory. Below we describe another type of Linear Programming problem, Binary Integer Linear Programming.

For these problems, we have the additional restriction that each variable $x_j \in \{0, 1\}$. There are many situations where we would like to use binary variables. For example, we might take as our binary variables $x_j = x_{t,p,c_u,c_v}$, which are 1 if at time t plane p leaves from city c_u to city c_v, and 0 otherwise. Assigning values to the x_{t,p,c_u,c_v} is equivalent to designing an airline schedule; we would then maximize revenue, which would be a function of the routes flown and demand and ticket costs for those routes.

Binary Integer Linear Programming is, of course, a specific example of a more general problem, namely Integer Linear Programming. Much of the difficulty of the subject stems from the fact that a problem may have optimal real solutions and optimal integer solutions, but the optimal integer solutions need not be close to the optimal real solutions. To see this, consider the knapsack problem (we follow the presentation in [Fr], pages 127–128).

Imagine we have a knapsack that can hold at most 100 kilograms. There are three items we can pack. The first weighs 51 kilograms and is worth \$150 per unit; the second weights 50 kilograms and is worth \$100 per unit; the third weighs 50 kilograms and is worth \$99 per unit. The goal is to pack as much as we can in the knapsack and maximize the value. Thus if x_j is the amount of the j^{th} item, we have the constraint

$$51x_1 + 50x_2 + 50x_3 \ \leq \ 100, \quad x_j \geq 0 \tag{49}$$

and we want to maximize

$$150x_1 + 100x_2 + 99x_3. \tag{50}$$

If we allow the x_j's to be real numbers, the optimal answer is $x_1 = 100/51$ and $x_2 = x_3 = 0$; the value of the knapsack is about \$294.12. The solution can be understood as the first product is the best value per kilogram, and as we are free to take non-integral amounts, we just take the first product. If we require the x_j to be integers, the optimal solution is $x_2 = 2$ and $x_1 = x_3 = 0$; the value of the knapsack is \$200. Thus not only is the optimal value significantly different when the x_j's are integers, but the answer is very different (rather than almost 2 units of the first item and none of the second, we have 2 units of the second and none of the others).

We give an example of the type of problems amenable to Binary Integer Linear Programming. Imagine that we are the manager of an Activity Hall or Community Center. There are various activities that we can schedule, and many constraints (we shall only give a few constraints, and leave it as an exercise to the reader to decide upon additional constraints which we, as the manager, may want to impose).

We shall consider binary random variables $x_j = x_{tar}$, where

$$x_{tar} \ = \ \begin{cases} 1 & \text{if at time } t \text{ we start activity } a \text{ in room } r \\ 0 & \text{otherwise.} \end{cases} \tag{51}$$

We assume

$$\begin{aligned} t &\in \{0,\dots,T\} \\ a &\in \{1,\dots,A\} \\ r &\in \{1,\dots,R\}. \end{aligned} \tag{52}$$

Thus we assume the activity hall opens at time 0 and closes at time T; perhaps each time period represents 5 minutes or 10 minutes. We assume there are A different activities that could be scheduled (perhaps they are bridge, chess, a movie, a lecture, and so on); note different activities might run for different amounts of time. Finally, we assume there are R rooms.

We need certain inputs:

1. Let D_{tar} be the number of people who would go to activity a if it starts in room r at time t. It is likely that for some activities the demand might be room independent; however, even in this case it is likely that it would depend on the time t. Further (we will not get into this now), it is natural to ask whether or not the demand of one activity depends on what other activities are occurring at the same time.

2. Let f_{tar} be the fee someone must pay to start activity a at time t in room r. It is likely that f_{tar} does not depend on the room, but it is easy to keep the added flexibility.

3. Let c_{tar} be the capacity of room r being used for activity a starting at time t. It is possible that the capacity depends on the activity (if we use a room for a basketball game we would probably have fewer people than if we used it for a chess tournament).

4. Let L_{tar} be the number of time units that it takes for activity a to finish, given that it started at time t in room r.

Our goal is to maximize revenue:

$$\max \sum_{t=0}^{T} \sum_{a=1}^{A} \sum_{r=1}^{R} \min(D_{tar}, c_{tar}) f_{tar} x_{tar}. \tag{53}$$

The above formula is fairly obvious; the only thing we must be careful about is that we cannot have more people in a room than it is capable of handling (hence the $\min(D_{tar}, c_{tar})$ factor). There are many possible constraints which we, as the manager, may want to impose. We list merely a few.

- At any time, in any room, at most one activity is running:

$$\forall t, r: \ \sum_{a=1}^{A} \sum_{t'=\max(t-L_{tar}+1,0)}^{t} x_{t'ar} \ \leq \ 1. \tag{54}$$

- At any time, each activity is running in at most one room:

$$\forall t, a: \ \sum_{r=1}^{R} \sum_{t'=\max(t-L_{tar}+1,0)}^{t} x_{t'ar} \ \leq \ 1. \tag{55}$$

If necessary, we might want to label activities such as bridge-1, bridge-2.

- We cannot have any activities running after the activity hall closes at time T:

$$\sum_{a=1}^{A} \sum_{r=1}^{R} \sum_{t=\max(T-L_{tar}+1,0)}^{T} x_{t'ar} \ = \ 0. \tag{56}$$

- Every b time blocks from T_s to T_e, some activity starts in some room:

$$\forall t'' \in \{0, \dots, T_e - T_s\} : \quad \sum_{t'=t''+T_s}^{\max(b-1+t''+T_s,T)} \ \sum_{a=1}^{A} \sum_{r=1}^{R} x_{t'ar} \ \geq \ 1. \tag{57}$$

The last constraint is of a different nature than the previous. The first three must be true for a schedule to be valid; the last is a natural constraint that a manager might wish to impose, but is not needed for a schedule to be realistic. The last constraint ensure that, if someone walks into the activity hall, they need not wait more than b time units before an activity will start.

Again, there are many additional constraints that a manager may wish to impose. We content ourselves with the above discussion, which hopefully highlights the utility and flexibility of such models.

Finally, it is not enough to model a program as a Binary Integer Linear Program; we need to be able to solve such models *quickly*. We encourage the reader to consult the literature for more on the subject.

Exercise 2.15. *Consider the above scheduling problem for the activity hall. Assume (i) each time block is 15 minutes and the hall is open for 10 hours a day; (ii) there are 10 different activities, each activity can be started at any time in the day; (iii) there are 12 rooms in the hall. How many binary integer variables x_{tar} are there? How many equations?*

Exercise 2.16. *Assume the activity hall has a concession stand. Let r_{tar} be the amount of concession revenue the activity hall earns from a person who starts activity a at time t in room r. What is the new revenue function? Note that, once we start adding additional terms to reflect concession revenue, other issues arise. For example, there is now the cost of staffing the concession stand, and perhaps the amount of help hired is a function of the number of customers.*

3 Versatility of Linear Programming

In this section we show how linear programming is able to handle a variety of constraints and situations. Explicitly, we describe how to handle

- Binary Indicator Variables;
- Or Statements;
- If-Then Statements;
- Truncation;
- Minimums and Maximums;
- Absolute Values.

Throughout this section, we assume *every* expression $A, B, \dots$ is bounded by N. For definiteness, we might write things such as N_A to show the constant depends on A (A may either be one of our variables, or a linear combination of variables). The fact that A is bounded by N_A means $|A| \le N_A$ or $-N_A \le A \le N_A$. We also assume each quantity is discrete, and the smallest non-negative unit is δ. This means the possible values attained are $0, \pm\delta, \pm 2\delta, \dots$.

3.1 Binary Indicator Variables

Theorem 3.1. *Given a quantity A, the following constraints ensure that z_A is 1 if $A \ge 0$ and z_A is 0 otherwise:*

1. $z_A \in \{0, 1\}$.
2. $\frac{A}{N_A} + \frac{\delta}{2N_A} \le z_A$.
3. $z_A \le 1 + \frac{A}{N_A}$.

Proof. The first condition ensures that z_A is a binary indicator variable, taking on the values 0 or 1. The second condition implies that if $A \ge 0$ then $z_A = 1$; if $A \le 0$ this condition provides no information on z_A. The third condition implies that if $A < 0$ then $z_A = 0$; if $A \ge 0$ this condition provides no information on z_A. □

3.2 Or Statements

Often we want one of two constraints to hold. There is the exclusive or (exactly one of two holds) and the inclusive or (both may hold, but at least one holds).

Theorem 3.2 (Exclusive Or)**.** *The following constraints ensure that $z_A = 1$ if $A \ge 0$ or $B \ge 0$ but not both, and $z_A = 0$ otherwise:*

1. $z_A \in \{0, 1\}$.
2. $\frac{A}{N_A} + \frac{\delta}{2N_A} \le z_A$.
3. $z_A \le 1 + \frac{A}{N_A}$.
4. $z_B \in \{0, 1\}$.
5. $\frac{B}{N_B} + \frac{\delta}{2N_A} \le z_B$.
6. $z_B \le 1 + \frac{B}{N_B}$.
7. $z_A + z_B = 1$.

Proof. The first three conditions ensure z_A is 1 if $A \ge 0$ and 0 otherwise; the next three ensure z_B is 1 if $B \ge 0$ and 0 otherwise. The last condition ensures that exactly one of z_A and z_B is 1 (and the other is 0). For example, if $z_A = 0$ then condition 2 implies that $A < 0$, and if $z_B = 1$ then condition 6 implies that $B \ge 0$. □

Theorem 3.3 (Inclusive Or)**.** *The following constraints ensure that $z_A = 1$ if $A \ge 0$ or $B \ge 0$ (and possibly both are greater than or equal to zero), and $z_A = 0$ otherwise:*

1. $z_A \in \{0, 1\}$.
2. $\frac{A}{N_A} + \frac{\delta}{2N_A} \le z_A$.
3. $z_A \le 1 + \frac{A}{N_A}$.
4. $z_B \in \{0, 1\}$.
5. $\frac{B}{N_B} + \frac{\delta}{2N_A} \le z_B$.
6. $z_B \le 1 + \frac{B}{N_B}$.
7. $z_A + z_B \ge 1$.

Exercise 3.4. *Prove the above theorem.*

3.3 If-Then Statements

Theorem 3.5 (If-Then)**.** *The following constraints allow us to program the statement: IF $(A < 0)$ THEN $(B \ge 0)$.*

1. $z \in \{0, 1\}$.
2. $N_A z \ge A$.
3. $A + (1 - z)N_A \ge 0$.
4. $B \ge -zN_B$.

Proof. If $A > 0$, the second constraint makes $z = 1$. The third constraint is trivially satisfied (as $A > 0$), and the fourth constraint becomes $B \geq -N_B$, which is trivially satisfied (we are assuming $|B| \leq N_B$).

If $A < 0$, the third constraint is satisfied only when $z = 0$. The fourth constraint now becomes $B \geq 0$.

If $A = 0$, the second and third constraints are always satisfied. Taking $z = 1$ we see the fourth is satisfied. $\square$

Cost: If-Then can be handled by adding one binary variable and three constraints (four if you count the binary declaration). We could also do IF $(A < A_0)$ THEN $(B \geq 0)$ just as easily.

3.4 Truncation

Given an integer variable or expression X, we show how to add constraints so a variable Y equals X if $X \geq X_0$, and $Y = 0$ otherwise. We constantly use the If-Then constraints.

Let $z \in \{0, 1\}$. IF $(X < X_0)$ THEN $z = 0$. IF $(X > X_0 - \frac{\delta}{2})$ THEN $z = 1$. *We can do this: take $B = z - \frac{1}{2}$ in the If-Then section. Note the $-\frac{\delta}{2}$ allows us to re-write the IF condition as $X \geq X_0$.*

The following three constraints finish the problem.

1. $(Y - X) + N_X(1 - z) \geq 0$
2. $Y - N_X z \leq 0$
3. $0 \leq Y \leq X$

If $z = 0$ (ie, $X < X_0$), the first constraint holds. As Y is non-negative, the second constraint forces $Y = 0$, and the third holds.

If $z = 1$ (ie, $X \geq X_0$), the first forces $Y \geq X$ which, combined with the third, forces $Y = X$. As $Y \leq X \leq N_X$, the second constraint holds.

3.5 Minimums and Maximums

Theorem 3.6 (Minimums). *The following constraints ensure that $Y = \min(A, B)$:*

1. *$Y \leq A$.*
2. *$Y \leq B$.*
3. *$Y = A$ OR $Y = B$.*

Proof. The first condition forces $Y \leq A$ and the second $Y \leq B$. Without loss of generality, assume $A \leq B$. The third condition says either $Y = A$ or $Y = B$ (or both). If $Y = B$ this contradicts the first condition. Thus $Y \leq A$ is improved to $Y = A$. $\square$

Theorem 3.7 (Maximums). *The following constraints ensure that $Y = \max(A, B)$:*

1. $Y \geq A$.
2. $Y \geq B$.
3. $Y = A$ *OR* $Y = B$.

Exercise 3.8. *Prove the above theorem.*

3.6 Absolute Values

Theorem 3.9. *The following constraints ensure that $X = |A|$:*

1. $A \leq X$.
2. $-A \leq X$.
3. $X \leq A$ *OR* $X \leq -A$.

Proof. The first two constraints force X to be a non-negative number, of size at least $|A|$. We just need to make sure $X \not\geq |A|$.

For the third constraint, if $A = 0$, the two or clauses are the same, and $X = 0$. If $A \neq 0$, as X is non-negative it can only be less than whichever of A and $-A$ is non-negative. □

One application is

Theorem 3.10. *Assume $|c^T x + b| \leq M$ for all x that are potential solutions of the Integer Programming Problem, and assume c, x and b are integral. We may replace a term $|c^T x + b|$ with y by introducing three new variables (y, z_1 and z_2) and the following constraints:*

1. $z_1, z_2 \in \{0, 1\}$.
2. $c^T x + b \leq y$
3. $-(c^T x + b) \leq y$
4. $0 \leq y$, y *is an integer*
5. $y \leq M$
6. $y - (c^T x + b) \leq 2 z_1 M$
7. $y + (c^T x + b) \leq 2 z_2 M$
8. $z_1 + z_2 = 1$.

Exercise 3.11. *Prove the above theorem.*

4 Removing Quadratic (and Higher Order) Terms

As the name implies, Linear Programming is about linear constraints and minimizing (or maximizing) linear objective functions; however, there are generalizations to quadratic constraints or objective functions. To describe these techniques would take us too far afield; however, we can quickly show how some of these constraints may be linearized.

4.1 Problem Formulation

For definiteness, we shall consider the following example. We are the owner of a movie theater, and our goal is to schedule movies. Let us say the time slots are in 10 minute increments (labeled $0, 1, \dots, T$), there are M movies (labeled $1, \dots, M$) and there are S screens (labeled $1, \dots, S$). We let

$$x_{tms} = \begin{cases} 1 & \text{if at time } t \text{ we start movie } m \text{ on screen } s \\ 0 & \text{otherwise.} \end{cases} \tag{58}$$

This problem is similar to the activity hall where we were the manager (see §2.5). We now allow certain polynomial non-linearities in the objective function which we are trying to maximize, and discuss how to linearize them. Although we are concentrating on non-linearities in the objective function (the revenue for a given schedule), the method is identical for removing such non-linearities from the constraints.

For example, in the case of movies, let's fix a time t and consider $m = m_1 + m_2$ movies. For convenience, we assume our two sets of movies are the first m_1 movies and then the next m_2 movies, though the general case proceeds similarly.

We introduce some new variables:

$$y_{tm} = \begin{cases} 1 & \text{if movie } m \text{ is playing at time } t \\ 0 & \text{otherwise.} \end{cases} \tag{59}$$

We may have corrections to the objective function, depending on which movies are being shown. For example, consider the case when $m_1 = m_2 = 1$. Thus each set contains just one movie. Imagine these two movies are *both* action blockbusters. It is reasonable to assume that the demand for each movie is affected by whether or not the other movie is playing. For example, perhaps many people go to a theater without having already chosen the movie they wish to see; perhaps they are just in the mood to see an action movie. If only one of these two movies is playing at a certain time, then anyone interested in an action movie *must* see that film; however, if both are playing then the customer has a choice. Thus we want to penalize the demand for an action movie if another action movie is playing at the same time.

Consider the polynomial

$$\begin{aligned} p(x) &= \prod_{m=1}^{m_1} y_{tm} \prod_{m=m_1+1}^{m_1+m_2} (1 - y_{tm}) \\ &= \begin{cases} 1 & \begin{array}{l}\text{if at time } t \text{ movies } m_1 \text{ through } m_2 \text{ are being shown} \\ \text{and movies } m_1+1 \text{ through } m_1+m_2 \text{ are not being shown;}\end{array} \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{60}$$

By considering all possible polynomials of this form, we can handle any m_1-tuple of movies playing and m_2-tuple of movies not playing. Note that every variable occurs to either the zeroth or first power: as $y_{tm} \in \{0, 1\}$, $y_{tm}^n = y_{tm}$ for any integer $n \geq 1$. ***This is an extremely useful consequence of being a binary variables!***

Our goal is to replace terms in the Objective Function of the form $-\text{Const} \cdot p(x)$ *with linear terms, possibly at the cost of additional variables and constraints.*

In our example, these $m_1 + m_2$ movies compete with each other for demand, and we must adjust the demand of each movie based on the competition concurrently screened.

4.2 Linearizing Certain Polynomials

Say we have a term $\prod_{m=1}^{m_1} y_{tm} \prod_{m=m_1+1}^{m_1+m_2} (1 - y_{tm})$. This is 1 or 0. Hence we can introduce a new binary variable δ_t equal to the above.

Thus $\delta_t = 1$ if the first m_1 variables y_{tm} are 1 (the first m_1 movies are on at time t) and the last m_2 variables y_{tm} are 0 (the next m_2 movies are not on at time t), and $\delta_t = 0$ otherwise. We can replace the product with δ_t as follows:

1. $\sum_{m=1}^{m_1} y_{tm} + \sum_{m=m_1+1}^{m_1+m_2} (1 - y_{tm}) - (m_1 + m_2)\delta_t \geq 0$
2. $\sum_{m=1}^{m_1} y_{tm} + \sum_{m=m_1+1}^{m_1+m_2} (1 - y_{tm}) - \delta_t \leq m_1 + m_2 - 1$.

If the first m_1 variables y_{tm} are 1 and the next m_2 variables y_{tm} are 0, then condition 1 doesn't constrain δ_t, but condition 2 forces $\delta_t = 0$ (as desired). If either one of the first m_1 variables y_{tm} is 0 or one of the last m_2 variables y_{tm} is 1, the first condition forces $\delta_t = 0$ and the second condition doesn't constrain δ_t. Therefore these two conditions encode δ_t linearly!

4.3 Example: Applications to Scheduling

Initially, we assumed the demand of a movie was independent of which movies were concurrently being shown. For simplicity we only partially worry about what screen a movie is shown on.

Let's assume movies compete for demand if they are shown within a certain amount of time (T_0) of each other. As always, x_{tms} is 1 if we start showing movie m at time t on screen s, and 0 otherwise.

We initially have terms like $\min(D_{tm}, C_s)x_{tms}$ in the revenue function, where D_{tm} is the demand for movie m at time t (we assume all movie demands are independent) and C_s is the capacity of screen s. We want to modify this to allow movies to compete with each other for demand.

We must subtract off corrections (to the demands) based on what movies are starting around the same time. One has to be a little careful with wrap-around time effects, but this is just meant as a rough sketch. Define

$$y_{tm} \;=\; \sum_{s=1}^{S} \sum_{t'=t}^{t+T_0} x_{tms}. \tag{61}$$

Thus y_{tm} is 1 if we start movie m on any screen between t and $t + T_0$, and 0 otherwise. We then define polynomials

$$p(x) \;=\; \prod_{m=1}^{m_1} y_{tm} \prod_{m=m_1+1}^{m_1+m_2} (1 - y_{tm}) \tag{62}$$

as before, and multiply by a suitable constant which will include the loss of revenue from *all* the movies. This might not be the best way to go. It might be better to modify the demands of each individual movie. This would lead to terms such as

$$\min\left(D_{tm}x_{tms} - \text{Const} \cdot p(x), C_s x_{tms}\right). \tag{63}$$

Of course, the minimum function isn't immediately in Integer Linear Programming, but it is trivial to handle (see §3.5):

1. $Y \leq D_{tm}x_{tms} - \text{Const} \cdot p(x)$
2. $Y \leq C_s x_{tms}$
3. $Y \geq 0$, Y is an integer

4.4 Summary

As remarked, we have only touched the beginning of a very important generalization of Linear Programming. It is important to analyze the *cost* of linearizing our problem, specifically, for real world problems can the linearized problems be solved (or approximately solved) in a reasonable amount of time?

We are reminded of a quote from Abraham Maslow, who remarked that if all one has is a hammer, pretty soon all problems look like nails. Once we know how to do and solve Linear Programming problems, it is tempting to convert other problems to Linear Programming problems. While this will be a reasonable solution in many situations, there are additional techniques that are better able to handle many of these problems.

References

[Da] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, 1963.

[Fr] J. Franklin, *Mathematical Methods of Economics: Linear and Nonlinear Programming, Fixed-Point Theorem*, Springer-Verlag, New York, 1980.

[St] G. Stigler, *The cost of subsistence*, The Journal of Farm Economics, 1945.