Linear Programming[*]

Graeme Taylor

January 29, 2006

The theory of linear programming could reasonably be considered the most important single idea in Operational Research- a branch of mathematics concerned with finding (with proof!) optimal strategies or solutions to problems from economics, management and industry. As the simplest model, it is often not appropriate to real world scenarios, but still provides considerable insight into the subtleties of modelling and optimisation. Study of Operational Research at the level of undergraduate mathematics therefore tends to be based entirely on linear programming and techniques related to the Simplex Method, the classic approach to solving linear programming problems.

To the modern ear, the name may suggest that linear programming is something to do with computers. However, programming in this context refers to programmes of events- managing materials, time, money and work- as opposed to computer programming. Certainly, however, computers are ideal for the donkey work of applying the iterative algorithms developed to tackle linear problems, which tend to call for vast amounts of routine calculations and manipulation of matrices.

Examples of linear programming problems

A problem in mathematical terms

Maximise the function $Z = 15x + 20y$ subject to constraints

$$5x + 8y = 16000$$

$$5x + 4y = 14000$$

$$x + 3y = 5000$$

$$x, y > 0$$

[*]First appeared on Everything2, at *http://www.everything2.com/index.pl?node_id=1783002*

Considering your mathematical background, you might consider this the most well-posed problem of the three I'm going to give, or complete gibberish. For the benefit of the latter, let's consider how such a formulation might arise.

A (toy) problem in plain english

> A soft drinks company produces 3 different brands by mixing blackcurrant and orange juice (sounds awful). To make a litre of the first drink, which sells for 1.50 profit, requires 0.3 litres of orange and 0.1 of blackcurrant; for the second, sold at a profit of 2, 0.2 litres of each; and for the third, again retailing at a profit of 1.50, 0.1 litres of orange, 0.2 of blackcurrant. It is obviously desirable to maximise the profit- given that 2500 litres of orange juice and 1800 litres of blackcurrant are available, how much of each brand should be produced to achieve this?

This problem describes in words a series of equations and inequalities; the task of answering it reduces to finding mathematical expressions like the first example and then applying a method to solve it. But linear programming can tackle more sophisticated problems, too-

A very vague question

> Having settled on an optimal blend of products, to what extent can the supply of blackcurrant or the profit on the third brand change before a better approach arises?

This question is of more practical interest, especially in the ever-changing world of commerce. Solutions that have to be regenerated from scratch after the slightest change, or that will be seriously flawed if the supplied data is slightly inaccurate, are of little use. Fortunately, once solutions are found, they can be subject to a vast amount of post-optimality analysis to answer such questions through the determination of ranges.

The Canonical Form linear programming problem

Any given problem can be written in many ways, with the translation from real world terms to mathematical formulae usually offering plenty of freedom for a budding operational researcher. As is often the case with mathematics, it turns out to be helpful to have a standard form for a problem, as then any techniques developed that work on an arbitrary problem of that form can be applied to any specific example with ease.

The canonical form Maximise $Z = \mathbf{c}^T\mathbf{x}$ subject to

$$A\mathbf{x} = \mathbf{b}$$
$$\mathbf{x} \geq 0$$
$$\mathbf{b} \geq 0$$

Here, A is a matrix, and letters in bold represent vectors- collections of numbers. The use of vectors allows the problem to scale to any size without altering the formulation, but at a price of abstraction. Those who are happy with the notation may be unhappy with the demands- there's not an inequality in sight! Fortunately, any linear programming problem can be written in this form, and illustrating how should help clear up concerns from either camp.

The canonical form of our first example

The vector $\mathbf{x}$ denotes the variables in the system- the things we can alter in our quest for profit. Our example had two variables, x and y, so $\mathbf{x} = (x, y)^T$ is a 2-vector (the transpose sign T simply means that I'd like to write this as a column- in plaintext, this is fiddly, so I don't). Further, we know what the vector $\mathbf{c}^T$ should be- it's (15,20), to ensure that multiplying out $\mathbf{c}T\mathbf{x}$ gives $(15, 20) \times (x, y)^T = 15x + 20y$ as desired.

The vector $\mathbf{b}$ denotes the constraints on the variables- the factors that keep our profit in check. Here, with three constraints, $\mathbf{b} = (16000, 14000, 5000)^T$. Notice that we treat the $\mathbf{x} \geq 0$ constraints separately from the more interesting constraints.

We're almost there- we just need a matrix of values A which , when multiplied by $\mathbf{x}$, equals $\mathbf{b}$. But we can't do this just yet, as we only know inequalities, not equalities. We cheat, by introducing what's known as a slack variable, to build equalities as follows:

$$5x + 8y + u = 16000$$

$$5x + 4y + v = 14000$$

$$x + 3y + w = 5000$$

$$u, v, w \geq 0$$

It should be clear that if the slacks are positive, then we can't possibly stray outside of our bounds. In an ideal world, all the slacks will end up as 0- we run right up to the limit of the constraints. However, it's rare that this can be achieved for each inequality- there's likely to be some slack somewhere. This becomes important in answering ranging questions like the third example above- bounds that have been precisely met are most likely to offer yet greater profit if they can be relaxed in some way.

The price of equality was that we ended up with five variables to describe a system where only two are relevant. Often you'll run out of letters, so convention is to work with variables x_1, x_2, x_3 and so on. For this small problem though, 6 letters isn't too confusing, and we've got a canonical form as follows: Maximise $z = 15x + 20y + 0u + 0v + 0w$ subject to

$$5x + 8y + 1u + 0v + 0w = 16000$$

$$5x + 4y + 0u + 1v + 0w = 14000$$
$$1x + 3y + 0u + 0v + 1w = 5000$$
$$x, y, u, v, w \geq 0$$

Just to make things crystal clear, that's a five variable system with $\mathbf{x} = (x, y, u, v, w)^T$, $\mathbf{b} = (16000, 14000, 5000)^T$, $\mathbf{c}^T = (15, 20, 0, 0, 0)$, and A given by

$$\begin{pmatrix} 5 & 8 & 1 & 0 & 0 \\ 5 & 4 & 0 & 1 & 0 \\ 1 & 3 & 0 & 0 & 1 \end{pmatrix}$$

Clearly, $\mathbf{x}, \mathbf{b} \geq 0$ as required.

The canonical form of our second example

If the above made sense, then you can skip to the general notes on obtaining canonical form in the section below. I don't adivse you skip those, as there are some other subtleties (surplus variables, sign restrictions) that the above problem didn't require but an arbitrary one might.

If however you'd like some practice, you can try to formulate the example of the drinks company. Supposing that x_i denotes the number of litres of product i produced, and introducing appropriate slacks, you should be able to come up with

$$\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)^T$$
$$\mathbf{c} = (1.5, 2, 1.5, 0, 0)^T$$
$$A = \begin{pmatrix} 0.3 & 0.2 & 0.1 & 1 & 0 \\ 0.1 & 0.2 & 0.2 & 0 & 1 \end{pmatrix}$$
$$\mathbf{b} = (2500, 1800)^T$$

If you couldn't get this yourself, write it out long-form and see if you at least agree that it matches the problem given.

Finding canonical form in general

A number of subtleties arise because of the stipulations on signs- we work with positive variables and positive constraints. Fortunately, if you have something different it can be rewritten in the desired form.

Minimisation problems

This is easily fixed- if you want to minimise $ax_1 + bx_2 + cx_3 + \dots$ then you can maximise $(-a)x_1 + (-b)x_2 + (-c)x_3 + \dots$ which will fit the canonical form since there was no sign condition for elements of c.

$\leq$ inequalities

As we saw above, a non-negative slack variable can be added to turn an inequality into equality: $x_1 \leq 5$ becomes $x_1 + x_2 = 5$, for instance. Note that you can't use any of the variables that already have a meaning in the problem, and that includes any previously added slacks- this is why in the first example we added in u,v and w rather than a single slack variable u. Further, you'll have to update the other equations to include zero lots of the slack, and the objective values c will typically be zero for any slack variable (since there's usually no value in doing nothing).

$\geq$ inequalities

In analogy with slacks, we can consider subtracting a non-negative surplus variable. So if $x_1 \geq 5$, then we can write this as $x_1 - x_2 = 5$. The concerns for slacks also apply for surpluses.

negative constraints

If you have something like $3x_1 + 2x_2 = -5$, then you'd have a negative entry for $\mathbf{b}$. Since these are forbidden, multiply the whole thing through by -1: $(-3)x_1 + (-2)x_2 = 5$ is equivalent and perfectly acceptable.

Non sign-restricted variables

It may be that a variable x_1 is not required to be positive, and the optimal solution may arise when it's negative. To get around this, split it into positive and negative parts by writing $x_1 = x_1^+ - x_1^-$ and solve for the two non-negative variables x_1^+, x_1^- instead.

Some Theory

After considerable effort we've got as far as a canonical form. Can we now solve the problem? The simple answer is yes- with appropriate software (from generic office kit like Excel to highly specialised solvers, via Computer algebra systems like Maple or MATLAB), you're now in a position to plug in and get the answer, absent of any understanding of what's going on. If you'd like to iterate by hand, you can also use the method outlined in the simplex method node- although until I've updated that writeup it only covers systems involving slacks, not surplus variables (which were presumably considered beyond A-level students). If you'd like to know what makes these things tick, hang around for the underlying theory; otherwise skip down a bit for some special cases of linear programming problems in disguise.

Notation

First, of course, some notation. There will be many values of $\mathbf{x}$ which meet out requirements; the challenge is finding the best one. To do that, of course, needs some sense of what makes a good solution.

Feasible solutions

Simply put, a solution that isn't feasible is no solution at all- it'd ask us to do the impossible. Thus our first collection of interest is the set F of *feasible solutions*:

$$F := \{x \mid Ax = b \text{ and } x \geq 0\}$$

If this means anything to you, it's worth noting that this is a convex set.

Basic solutions

For n variables we can treat the matrix A as n column vectors $\mathbf{a_i}$. However, if the number of constraints m is greater than the number of variables there won't be any choices to make, so certainly there's no interesting optimisation to be done. When $m < n$, however, we can try to make life easier for ourselves- if we can find m linearly independent columns of A, then we can get a unique solution involving just m of the variables, with the rest assumed to be zero. This (non obvious) approach is equivalent to finding extreme points of F- and these in turn emerge as the candidates for optimality. Thus:

> $\mathbf{x^0} = (x_1^0, x_2^0, \ldots, x_n^0)$ is a *basic solution* if $A\mathbf{x^0} = \mathbf{b}$ and the set $\{\mathbf{a_i}\}$ for those i corresponding to x_i^0 non-zero is linearly independent.

A *basic feasible solution* is a basic solution with non-negative entries (i.e., a basic element of F).

Optimal Solutions

This is the desired property; a solution such that no other valid solution can give rise to a greater valuation on the objective function z. Symbolically, this is:

$$\mathbf{x}^0 \text{ is optimal if } \mathbf{x}^0 \in F \text{ and } \forall \mathbf{x} \in F,\ \mathbf{c}^T\mathbf{x}^0 \geq \mathbf{c}^T\mathbf{x}$$

This leads us to the central result:

The Fundamental Theorem of Linear Programming

> If the canonical form linear programming problem has a finite optimising solution, then there is a basic feasible solution which is optimal.

This is saying something very powerful- that of the myriad optimal solutions, there will always be a basic feasible one, which (obviously) does the job just as well. This immediately gives us a naive solution method- test all the basic feasible solutions and see which one is best. However, a better approach is offered by the simplex method- whilst in the worst case it reduces to this naive testing, ordinarily it will quickly home in on optimal solutions. This is because, given an initial basic feasible solution it is guaranteed (under certain conditions of non-degeneracy, although even that can be resolved) to move to a strictly better solution, terminating when this is not possible. Thus, upon termination, one can be certain that an optimal solution has been reached, in which some (often many) of the variables are zero (hopefully slacks).

Special Cases

Any linear programming problem written in canonical form can be tackled by the Simplex method (although in some cases a different approach, Karmarkar's algorithm, is preferred), some problems have additional structure that can be exploited for more elegant solution methods. I'll briefly outline some of those (without solution) here.

The Transportation Problem

The motivation for this problem is to minimise the cost of transferring goods from sources to destinations, in accordance with constraints on supply at each source and demand at each destination. In the balanced problem, we consider m sources with varying availabilities a_i and n destinations each requiring some amount b_j, with $\sum a_i = \sum b_j$. Transporting 1 unit from source i to destination j is assumed to incur a cost of c_{ij}, and we have a linear programming problem in mn variables; assuming assignments x_ij we also generate $m + n$ constraints through supply/demand considerations.

This turns out to be more easily manipulated as a special table of values than via the simplex algorithm, and an algorithm exists which, as expected, iteratively improves upon the objective by moving from one basic feasible solution to another. Furthermore, the technique can be modified to include unbalanced problems or to exclude routes through addition of dummy sinks and sources and 'pricing out' certain entries. A wide class of problems can be tackled as transportation problems- from scheduling production/storage to a mathematical justification of monogamy!

Network Flow

Given a network- be it rail infrastructure, internet routers or tasks on an assembly line- and capacities of connections between each node, what is the optimal flow through the network? Where do bottlenecks occur? These again are linear programming problems. The Ford-Fulkerson algorithm exploits the maximum flow-minimum cut theorem, which is equivalent to a powerful result in linear programming known as duality theory.

The assignment problem

Given a list of jobs and list of candidates, each with varying competency, the assignment problem is to match workers to jobs in such a way as to maximise the sum of ratings (standard assignment) or the minimum competency (bottleneck assignment). Here we are spoilt for choice: this could be formulated as a very complicated simplex problem or a fairly complicated transportation problem but the best technique turns out to borrow from network flow theory as well.