Maximize Profit

MP in Action

The Newsletter of Mathematical Programming in Industry and Commerce

May 1995

How to Build a Mathematical Programming Model

by Robert Simons

This month's article continues an occasional series on aspects of the practice of Mathematical Programming.

This article is intended to provide guidance on three points:

- is my problem likely to be amenable to Mathematical Programming?
- how should I set about formulating my model?
- what should I include and what should I leave out?

It does not attempt to address issues of what software is most appropriate, on which guidance was offered in November 1994's and January 1995's *MP in Action*. Nor is it a substitute for a training course or a textbook such as H.P.Williams' *Model Building in Mathematical Programming* (Wiley, 1993). Rather it is intended as practical guidance for those setting out to tackle a problem.

Identifying the Problem

Mathematical Programming is one of a number of OR techniques. Its particular characteristic is that the *best* solution to a model is found *automatically* by optimization software. An MP model answers the question "What's best?" rather than "What happened?" (statistics), "What if?" (simulation), "What will happen?" (forecasting) or "What would an expert do and why?" (expert systems).

Being so ambitious does have its disadvantages. Mathematical Programming is more restrictive in what it can represent than other techniques. Nor should it be imagined that it really does find the best solution to the real-world problem. It finds the best solution to the problem *as modelled*. If the model has been built well, this solution should translate back into the real world as a good solution to the real-world problem. If it does not, analysis of why it is no good leads to greater understanding of the real-world problem.

Mathematical Programming

Mathematical Programming is the process of using mathematical models to help find good solutions to business problems. Its key feature is that the mathematical model is *optimized*. This finds better solutions than other techniques and leads to greater understanding of the problem by rigorously challenging the model's assumptions.

The essential characteristics of a problem for Mathematical Programming to be applied are:

- many potentially acceptable solutions
- some means of assessing the quality of alternative solutions;
- some interconnectedness between the variable elements of the system.

These are reflected in the essential components of an MP model:

- the values of the **decision variables**, which describe the solutions;
- the **objective function**, which measures the quality of solutions;
- the relationships between decision variables, or **constraints**.

Mathematical Programming is very suitable for problems involving blending, continuous flow processing, production and distribution, and strategic planning. It answers questions such as:

- how much?
- when?
- where?

The principles of MP also apply to problems involving logistics and scheduling but the processes of tackling such problems are more varied and a mixture of techniques is likely to be used, including MP, heuristics and special-purpose algorithms.

Formulating the Model

Much of the art of building an MP model revolves around deciding which aspects of a real-world problem should be included and which should not. In practice this is an iterative process. To start with, keep things simple. If in doubt, leave out. If the optimum solution from the resulting model is clearly wrong in the real world, add extra detail and try again.

When starting to formulate a model, it may be helpful to think in terms of the typical decision variables of an MP model. These are:

- buying (or importing from outside the system being modelled);
- making;
- moving from place to place;
- storing (or moving through time);
- selling (or exporting to outside the system being modelled).

Typical constraints are:

- availability (of materials, resources);
- capacity (of processes);
- quality (upper and lower limits);
- demand;
- material balance (within the system being modelled).

With these in mind, identify what data are available and construct the model to fit them. If there are some data which appear to be crucial but which are not available, consider how decisions are being made now. If judgements are being made based on estimates, try to obtain the estimates and use them. If estimates are not available, push ahead regardless and await the reaction to the results from your model. The supposed crucial factor may not matter very much, in which case you are better off without it. If the results of your model are nonsense, the explanation of why they are so should provide some guidance as to what to do.

Another aspect of the 'keep it simple' principle is 'keep it linear'. Most optimization procedures are based on Linear Programming, which is unique for its ability to find the global (as opposed to local) optimum quickly and efficiently. Many non-linear relationships can be represented reasonably well by linear approximations and the benefits of having reliable, completely automatic optimization procedures outweigh the disadvantages, at

least in the early stages of model building. A further benefit of Linear Programming is the automatic availability of sensitivity data in the form of marginal values. Likewise, even if certain decision variables must in practice take integer values, it is usually best to ignore this at first.

The Primacy of Data

When building an MP model the data really do define the problem. Once you have decided which data are relevant, the choice of decision variables and constraints is largely fixed. Each item of data must be used; and for each decision which the model has to make, there must be an item of data to direct the model's choice.

If there is an item of data which has not been used, try to identify how it is used and add the necessary relationship. Follow this principle and the only constraints which you run the risk of omitting are material balances. Fortunately, these do not cause infeasibility in the resulting model, which is difficult to diagnose, but unboundedness or a solution which is, quite literally, too good to be true (because the model can effectively make things for nothing).

The need for items of data to direct each of the decisions which the model makes is more subtle. Consider a two-stage distribution process in which identical goods are shipped from several factories to several depots and thence on to customers. The cost of the goods varies between factories and there are separate costs for each leg of the distribution process. It is wrong to define as the decision variables the quantity of goods to be shipped from each factory through each depot to each customer. Instead you should define separate decision variables for the quantity of goods to be distributed from each factory to each depot and from each depot to each customer.

Defining a single set of two-stage decision variables purports to identify for each customer the factory where those goods were manufactured. From this one might be tempted to assess the profitability of supplying each customer, and even whether to continue to do so. Yet what data do we have to drive such decisions? The goods from each factory are identical. If we consider a depot which is supplied from two factories the costs of goods will differ depending on their source. The depot has to supply a number of customers. How do we decide which customers to supply with goods from each of the source factories? Without further information (such as that there are in practice differences between the goods from each factory and preferences among customers) *we simply have no basis for a decision.* Our model must reflect this.

When writing a formulation, record against each data item, decision variable and constraint the units in which that item is measured (e.g. $/tonne). Check that the units of all the terms (i.e. coefficient multiplied by decision variable) in each constraint are consistent with the stated units of that constraint. Pay particular attention to supposedly dimensionless data items, such as yield factors, and data with artificial units, such as quality indices. Yields may be expressed by mass (i.e. mass of output material as a fraction of mass of input material), by volume or by mol. If the associated decision variables are not measured the same way, conversion factors will be required.

Tackling Infeasibility

One of the greatest bugbears of practical MP work is infeasibility. You have put together a model with some data but the optimization package declares the resulting problem to be infeasible. What should you do? You know that the real problem isn't infeasible, so you have probably made a mistake in defining your formulation. Or has it been in specifying the data?

If you are lucky, the optimization package will identify a small number of constraints and bounds on decision variables which together make the entire problem infeasible. There are some free-standing infeasibility analysers which go further than this and identify a chain of relationships which define the infeasibility. These tools are all valuable.

But it remains true that an ounce of prevention is worth a pound of cure. First of all, the data should be validated before they reach the model so as to identify obvious gross errors. In practical applications this finds perhaps 80 - 90% of data errors before the optimization. Nonetheless, there are some inconsistencies which can slip through. And it is often the case that one only writes the data validation procedures once the model has been prototyped, so analysis of infeasibilities is required before this.

When you formulate your model you have an immense advantage over even the most sophisticated infeasibility analyser. *You understand your problem.* You know which constraints it makes sense to consider breaking. You can permit the model to violate such constraints *in extremis* by introducing extra decision variables in them known as explicit slacks, which are penalised heavily in the objective function. Thus quality constraints often carry explicit slacks while material balance constraints do not.

Some care is needed in specifying the penalty weights on the explicit slacks in the objective function so that constraints are violated in a "sensible" way when the data define an otherwise infeasible problem. Make the penalty weight directly proportional to the undesirability of violating the constraint and inversely proportional to the magnitude of a potential violation.

Another technique is to write constraints so that they retain their force even if other constraints are violated. For instance, quality constraints can be written so that they apply irrespective of how much material is produced. This means that the model cannot overcome a quality infeasibility by failing to meet demand.

Using these techniques it is possible largely to convert infeasibility into measured and meaningful violations of constraints which provide strong clues as to where any problems in data or in the formulation of the model lie.

Robert Simons is a consultant specialising in optimization and Secretary of the Mathematical Programming Study Group.

Mathematical Programming Study Group

The Mathematical Programming Study Group meets in London to advance understanding of Mathematical Programming. It is supported by the Operational Research Society and the British Computer Society.

Its next meetings will be at 18.30 on 13th June 1995 at the London School of Economics when Douglas Rogers (University of South Florida) will talk on Recursive Minimization and at 18.30 on 27th June 1995 at Friends House when Mike Trick (Carnegie Mellon University) will talk on Computational Voting Theory.

Printed and published by Eudoxus Systems Ltd
Optimum Solutions to Business Problems
Perth House Soulbury Road
Leighton Buzzard LU7 7RN U.K.
Tel: +44 1525 852660 Fax: +44 1525 852654 URL: www.eudoxus.com