Alistair Clark

Free modelling languages for linear and integer programming

Alistair Clark
Faculty of Computing, Engineering and Mathematical Sciences
University of the West of England
alistair.clark@uwe.ac.uk

Abstract

This paper illustrates the use of a mathematical programming language when teaching Linear and Integer Programming models on Operational Research courses. Such languages have often required expensive optimization software, but fortunately, the world-wide scope of the internet has put powerful free optimization tools within the reach of anyone with a PC connected even briefly to the internet. An effective example that the University of the West of England uses in its postgraduate MSc Statistics and Management Science is the NEOS server. This freely-accessible tool permits the submission of large-scale managerial optimization problems over the internet in a wide-variety of formats and sends the results back by email, often very quickly.

1. Introduction

On the MSc *Statistics and Management Science* at the University of the West of England, I teach a single-semester introductory *Operational Research* (OR) module to students from a wide range of undergraduate subject backgrounds. There is not a lot you can thoroughly cover in 12 weeks of teaching, so I tend to focus on my own speciality, linear and integer programming (LIP), with emphasis on modelling rather than algorithms. I teach the simplex merely conceptually, and then show how Excel can be used to solve small LIP models.

Spreadsheets are great for the classroom and/or small simple models, but are they appropriate for large, complex, changing models? The advantages of spreadsheets include the power and breadth of their functions for quantitative analysis, and their intuitive grid-like user interface with which many users are familiar and comfortable. Spreadsheets are omnipresent in many organizations, so there is already a large knowledge base upon which to draw. Specifically for OR, spreadsheets offer a multitude of resources such as dynamic recalculation and chart updating, statistical analysis, built-in optimisation algorithms (such as Excel Solver), programming languages (such as Excel's VBA), database connectivity, rapid application development with visual components, and the widespread availability of specialist "add-ins".

However spreadsheets have significant limitations when applied to OR analysis. It is easy and tempting to quickly create obscure and unintelligible models. Spreadsheets cannot easily represent OR models that are complex, or change frequently. They are also too slow to analyse or optimise models with very large amounts of data. Moreover, spreadsheets are notoriously prone to errors [1] which are frequently not obvious, creating a dangerous over-confidence in calculation results.

As a result it is often cumbersome and time-consuming to build, modify and maintain a large error-free spreadsheet model. These quality and effort concerns argue against the use of spreadsheets in prototyping and implementing complex models. An alternative that is faster, more flexible and less error-prone is to use a modelling language.

"Quality and effort concerns argue against the use of spreadsheets in prototyping and implementing complex models. An alternative that is faster, more flexible and less error-prone is to use a modelling language."

2. Modelling languages for mathematical programming

There are several modelling languages and systems for mathematical programming, including AMPL [**www.ampl.com**], AIMMS [**www.aimms.com**], GAMS [**www.gams.com**], XPress-MP [**www.dashoptimization.com**], and OPL [**www.ilog.com/products/oplstudio**]. I will focus on AMPL (A Mathematical Programming Language), used for teaching and research at several UK universities, including my own and the London School of Economics.

At the University of the West of England, after about a months' teaching and a few AMPL lab sessions, I hand out a postgraduate coursework that requires each student to choose an application area, survey the academic literature for LIP models in the area, synthesise a simple model typical of the area, and then implement it in AMPL, first on a small instance using the free student edition, and then on a larger instance using the free online NEOS Optimisation Server, as shown below. Although the coursework is demanding and requires some instructor support, feedback from students is generally very positive. They come out with CV-able AMPL skills, and many go on to use it in their MSc dissertation, honing their proficiency in modelling and optimization systems.

2.1 The AMPL modelling language

An effective way to start teaching a modelling language is to use a simple example. Consider a company that manufactures two products, Xingu and Yoba, at its three plants in Andradas, Betim and Cambuí. The data in Table 1 is available:

	Hours needed per batch		Hours available
	Xingu	Yoba	
Andradas	1	0	4
Betim	0	2	12
Cambuí	3	2	18
Profit/batch	$3,000	$5,000	

Table 1 – Simple example data to start teaching a modelling language

The problem of deciding how many Xingu and Yoba to produce with the objective of maximizing total profit can be formulated as a linear programme (LP) as follows:

Decision Variables:

x_1 = number of batches of Xingu produced

x_2 = number of batches of Yoba produced

Objective Function:

Maximize $3x_1 + 5x_2$ [total profit in $000s]

Constraints:

$x_1 \leq 4$ [Andradas capacity]

$2x_2 \leq 12$ [Betim capacity]

$3x_1 + 2x_2 \leq 18$ [Cambuí capacity]

$x_1, x_2 \geq 0$ [non-negativity constraints]

In AMPL (as in most modelling languages), data is separated from the model whereas they are mixed together in the formulation above. Thus the AMPL model for the above formulation is generic:

```
set Plants;
set Products;

param Avail {Plants};
param UnitProfit {Products};
param Usage {Plants, Products};

var Amount {Products} >= 0;

maximize Profit:
sum {j in Products}
  UnitProfit[j] * Amount[j];

subject to Capacity {i in Plants}:
sum {j in Products}
  Usage[i,j] * Amount[j]
<= Avail[i];
```

The model above declares the necessary indices (`set`), and then the indexed data structures (`param`) and decision variables (`var`). The LP's objective function is declared, called Profit and specified accordingly. Take note of the sum function. Finally, a set of indexed constraints called Capacity is specified, making use of the sum function.

Observe the complete absence of instance data in the AMPL model; it merely specifies the logical structure of the LP formulation. The model is supplied in a file on its own (for

example, product.mod). The data is supplied separately in another file (for example, product.dat):

```
data;

set Plants := Andradas Betim Cambuí;
set Products := Xingu Yoba;

param Avail :=
Andradas   4
Betim     12
Cambuí    18;

param UnitProfit :=
Xingu  3
Yoba   5;

param Usage: Xingu Yoba :=
Andradas       1    0
Betim          0    2
Cambuí
```

The AMPL solution run commands are specified in a third file (called, for example, product.run):

```
# load model file
model product.mod;
# load data file
data product.dat;
# use CPLEX to solve model
option solver cplex;
solve;    # solve the model
# display solution values
display Amount, Profit;
```

Note that the run file loads model and data files, specifies that the solver to use is Cplex, issues an instruction to solve the model, and finally displays the values of the decision variables Amount, and the resulting value of the objective function Profit. Any text after a # symbol is a comment (very useful for annotating a file) and so ignored by the AMPL processor.

AMPL can interface with a variety of optimization solvers for problems of the following types: Linear (simplex, interior or network), Quadratic (simplex or interior), Nonlinear (various), and Mixed Integer-Continuous (linear or nonlinear).

On Microsoft Windows, the run file is executed within AMPL as follows:

1. Create a batch text file called product.bat with the following single-line content:
   `ampl product.run > product.out`
2. Run the batch file product.bat by double clicking on it.
3. Examine the output of the run by opening the file product.out

The run output (in file product.out) for the above example is

```
CPLEX 10.0.1:
optimal solution;
objective 36
0 dual simplex iterations (0 in phase I)
Amount [*] :=
Xingu  2
Yoba  6;

Profit = 36
```

The output shows:

- that version 10.0.1 of the solver Cplex [**www.cplex.com**] was used to solve the model;
- that an optimal solution was obtained with objective value $36,000;
- that 0 dual simplex iterations were used in finding the solution; and
- the solution results outputted using the display command.

2.2 Increasing the instance size

It is now very simple to solve a larger instance of the same model with 5 plants and 6 products. The same model file product.mod is used unchanged, but the data file product.dat must be edited to represent the new problem instance:

```
data;

set Plants := Andradas Betim Cambuí
Diadema Embu;

set Products := Xingu Yoba Micos Nada
Pula Quenada;

param Avail :=
Andradas  4
Betim    12
Cambuí   18
Diadema  30
Embu     40;

param UnitProfit :=
Xingu    3
Yoba     5
Micos    2.4
Nada     1.2
Pula     3.5
Quenada  2.6;

param Usage: Xingu Yoba Micos Nada Pula
Quenada :=
Andradas  1 0 0 1.5 0 1.5
Betim  0 2 0 1.6 2.1 0
Cambuí  0 2.1 2 1.3 2 0
Diadema  0 2 2.1 0.8 2 0
Embu  1.1 0 2.2 0.7 1.9 0;
```

This gives the following output:

```
CPLEX 10.0.1:
optimal solution;
objective 48.48
2 dual simplex iterations (1 in phase I)
Amount [*] :=
Pula  0
Nada  0
Quenada  0
Xingu  4
Micos  2.7
Yoba  6;

Profit = 48.48
```

This shows that

- now 2 dual simplex iterations were used in finding the solution;
- Cplex obtained an optimal solution with objective value $48,480;
- but the number of Micos batches produced is fractional at 2.7.

2.3 Integer variables

To impose integer production values, edit AMPL model file `product.mod` so that the line:

```
var Amount {Products} >= 0;
```

becomes

```
var Amount {Products} integer >= 0;
```

The output in file `product.out` is now:

```
CPLEX 10.0.1:
optimal integer solution; objective 46.8
2 MIP simplex iterations
0 branch-and-bound nodes
Amount [*] :=
Pula  0
Nada  0
Quenada  0
Xingu  4
Micos  2
Yoba  6;

Profit = 46.8
```

which shows that:

- that 0 branch-and-bound nodes were used in finding the solution;
- the number of batches of all products is integer; and
- that Cplex obtained an optimal solution with objective value $46,800 [less profitable than the previous non-integer solution].

2.4 How to do it legally for free

The free student version of AMPL (available at **www.ampl.com**), can handle up to 300 variables and 300 constraints. Any attempt to run a model instance that exceeds these limits will result in an error message such as:

```
Sorry, the student edition is limited
to 300 variables and 300 constraints
and objectives (after presolve). You
have 656 variables, 332 constraints,
and 1 objective.
```

The full unlimited version of AMPL is expensive [although there are academic discounts for research and teaching], but there are free (legal) ways to get around this, as shown in the next sections, and in a recent review of non-commercial software for Mixed-Integer Linear Programming [2].

One possibility is to use the GNU Linear Programming Kit (GLPK) [**www.gnu.org/software/glpk**] which is free (and legal). It contains the GNU MathProg language, a subset of AMPL and also provides the GLPSol solver which, however, is *much slower* than Cplex. GLPK also has a set of routines organized into a callable library within the C programming language, but both MathProg and GLPSol can be used alone without using C.

2.5 The NEOS server

On the MSc *Statistics and Management Science* at the University of the West of England, I ask my students to solver larger instances using the NEOS Optimization server [**www-neos.mcs.anl.gov**]. Based at Argonne National Labs in the USA, the NEOS server is open and free to the public for the optimization of large models within certain time or solver-dependent iteration limits. A user submits three AMPL files online: the model, a data set, and a run file, as shown in Fig 1.

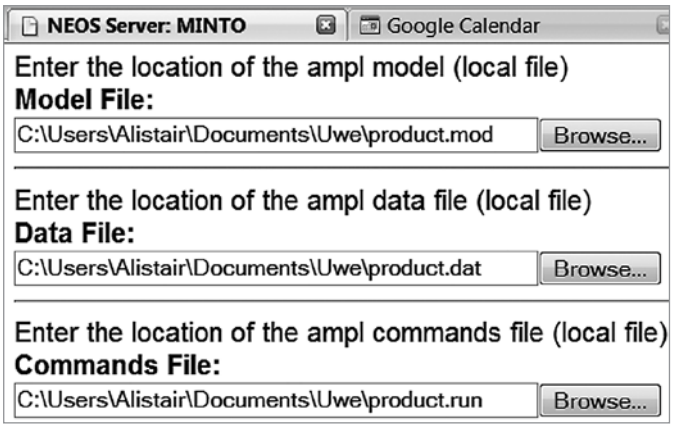

Fig 1 – Input of AMPL files to the NEOS server

The output appears on the screen (eventually) and, more reliably, is also emailed to you:

```
---- Begin Solver Output ----
You are using the solver mintoamp.
Executing AMPL.
processing data.
processing commands.

Presolve eliminates 1 constraint.
Adjusted problem:
6 variables, all integer
4 constraints, all linear; 14 nonzeros
1 linear objective; 6 nonzeros.

MINTO, a Mixed INTeger Optimizer  Version
3.1.0 (LINUX/COIN-OSI)

amount [*] :=
Armchair  0
   Chair  0
    Desk  0
    Door  4
   Table  2
  Window  6;

profit = 46.8
```

The NEOS server is very useful for research, teaching, student coursework, and prototyping if you work in an organization that does not have access to the commercial version of AMPL and its default (excellent) solver Cplex. NEOS offers a huge variety of solvers, but not Cplex. Not all its solvers all take input from AMPL, but one which does is Minto [**coral.ie.lehigh.edu/minto**], as used above.

2.6 COIN-OR

After prototyping an optimization model, a free operational version can be implemented that relies neither on commercial software nor the NEOS server, yet can easily take advantage of commercial solvers (such as Cplex), if available. This is achieved by using the open-source LP solver CLP [**www.coin-or.org/Clp**] through the Open Solver Interface (OSI) of the Computational Infrastructure for Operations Research (COIN-OR) project [**www.coin-or.org**], an initiative to spur the development of open-source software for the OR community.

The development effort, in the C programming language, can be substantial as the model matrix has to be specified element by element, a time-consuming task requiring detailed attention. Thus COIN-OR is not suitable for prototyping, but rather for stable models.

3. Conclusion

This paper has shown that powerful Linear and Integer Programming modelling languages and optimisation tools are available to universities, students, teachers and researchers, worldwide without having to purchase expensive software. They are within the reach of anyone with an internet connection. An effective example used at the University of the West of England is the AMPL on the NEOS server. This freely-accessible tool permits the submission of large-scale managerial optimization problems and shows the results onscreen and by email.

"Powerful Linear and Integer Programming modelling languages and optimisation tools are available to universities, students, teachers and researchers, worldwide without having to purchase expensive software. They are within the reach of anyone with an internet connection."

References

1. Finlay, P.N. and Wilson, J.M. (2000), A survey of contingency factors affecting the validation of end-user spreadsheet-based decision support systems, *Journal of the Operational Research Society*, Vol. 51(8): 949-958

2. Linderoth, J. T. and Ralphs, T. K. (2005), *Noncommercial Software for Mixed-Integer Linear Programming, in Integer Programming: Theory and Practice*, John Karlof (ed.), CRC Press Operations Research Series, pp 253-303, available online at webpage: http://coral.ie.lehigh.edu/pubs/files/jtl3_noncomm.pdf [Accessed 27 April 2007].