

MODELING AND TECHNOLOGY FOR MAKING QUANTITATIVE MANAGERIAL DECISIONS

Alistair Clark

*University of the West of England
Faculty of Computing, Engineering and Mathematical Sciences,
Bristol, BS16 1QY, England.
Alistair.Clark@uwe.ac.uk*

Paper presented at the 20th Scientific International Conference,
Hanoi University of Technology, Vietnam, 12-13 October 2006.

ABSTRACT

In recent years, powerful modeling languages such as AMPL (A Mathematical programming Language) have enabled Operational Research (OR) practitioners to rapidly develop prototype tools capable of modeling complex managerial decisions such as staff shift scheduling, or production & supply chain planning. However, such tools have often required expensive commercial optimization solvers that are sometimes beyond the financial reach of small companies and organizations, particularly in the low-income and emerging economies. Fortunately, the world-wide scope of the internet has put powerful free optimization tools within the reach of anyone with a modest PC and even a slow internet connection. An effective example that the University of the West of England uses in its postgraduate MSc in 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 (for example, AMPL), and sends the results back by email, often very quickly. The presentation will present examples showing just how beneficial such an approach can be for resource-poor organizations. It will also point to free (or nearly free) software tools capable of carrying out managerial quantitative analyses, including the humble-but-powerful spreadsheet.

1. INTRODUCTION

The application of Operational Research (OR), known as Operations Research in the USA, has the potential to radically enhance decision-making in organisations at the strategic, tactical and operational levels. This is particularly true when the decisions are quantitative or combinatorial in nature and involve the simultaneous consideration of many variables and constraints.

To emphasize the importance of OR, the North American Institute for Operations Research and the Management Sciences (INFORMS), the Association of European OR Societies (EURO) and the British OR Society have all been promoting OR to

business and the public sector through a joint publicity campaign [1]. Its target audience, however, is mainly executives in nations with more developed economies, and the campaign does not address the context of countries with emerging economies.

Emerging-economy countries differ very much between themselves, from the technologically advanced (e.g., Brazil, Chile, India, China) to the relatively deprived (e.g., West Africa). Brazil & Chile have developed Information and Computing Technology (ICT) sectors, a strong OR presence with specialist university researchers, sophisticated OR

projects in agro-business and industry [2, 3], and purchased access to state-of-the-art OR software. In contrast, the poorer emerging economies have less apparent demand for OR, a smaller OR presence with fewer university researchers, and limited access to ICT. For such countries, specialist OR software is often too expensive to buy and there is usually little or no local technical support in the country.

Thus the question to ask is: Are there less costly (or even free) software tools for OR that emerging economies can take advantage of? The answer turns out to “Yes”, to a surprising extent, as this paper now shows.

In section 2 we look at the use of spreadsheets in OR, and then proceed to demonstrate a powerful example of free specialist OR software in section 3. We discuss the implications and conclude in section 4.

2. OPERATIONAL RESEARCH AND SPREADSHEETS

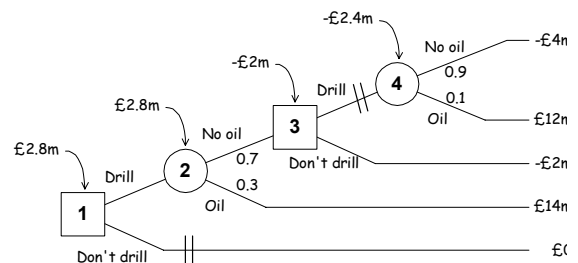
The use of spreadsheets such as Microsoft Excel is an increasingly popular way of applying OR approaches and techniques [4]. Their advantages include the power and breadth of their functions for quantitative analysis, and their intuitive grid-like user interface with which users are familiar and comfortable. Spreadsheets are omnipresent, being widely-used in many organisations and schools, so that there is already a large knowledge base upon which to draw. In many organisations, the most well known spreadsheet, Excel, is often already available and installed on a personal computer (PC), thus enhancing the transportability of spreadsheet models and lowering (or even zeroing) the costs of its use. There are even free lesser-known spreadsheets, such as OpenOffice’s *Calc* (on both Windows and Linux) and *Gnumeric & K-Office* (just on Linux). Google recently introduced a web-based spreadsheet [5] that can be simultaneously edited in real time by multiple users in different locations and stored online.

Specifically for OR, spreadsheets offer a multitude of resources: dynamic recalculation and chart updating, statistical analysis, built-in optimisation algorithms (such as Excel Solver), programming language (such as Excel’s VBA), database connectivity, rapid application development with visual components, and the widespread availability of specialist “add-ins”.

Thus much OR analysis can potentially be carried out with spreadsheets, such as project scheduling, simulation, decision trees, mixed integer linear programming, non-linear optimisation, and multi-criteria decision analysis. This capability has led to the concept *end-user modelling* [6,7] whereby the decision maker directly constructs a model, without the help of an OR specialist, in order to perform analysis and obtain insight.

2.1 Decision Tree Example

The decision tree below illustrates the possible decisions (square nodes) to be made by a company prospecting for oil in the face of uncertainty (circle nodes):



A manual analysis shows that the best EMV policy is to drill at 1st site, but if no oil is found, not to drill at 2nd site. A spreadsheet model of the decision tree and analysis can be constructed in a two-stage process:

1. working downwards, building up the structure of the tree. If at a square decision node, fill in as much as known in the first four columns. If at a round chance node, fill in as much as known in the first six columns. Use the Excel functions `=SUM(...)` for the Expected Values (EV) or `=MAX(...)` for the

Maximum EV in the 4th column, even when some of the cells from which they take their values are still empty.

- working back upwards, filling in those forgotten end-state payoffs or references to cells where Expected Values have been calculated. The answer will appear as soon as you get back to the top.

The completed spreadsheet (showing the underlying *Excel* formulae) might look like this:

	A	B	C	D	E	F
1		Oil reserves (m gallons)	16			
2		Profit per gallon (£)	1			
3		Cost of test drill (£m)	2			
4						
5	Node	Node Type	Branch (next node)	Value (£m)	Probability	Prob. X Value
6	1	Decision	Drill (2)	=F12		
7			Do not drill	0		
8			Maximum EV	=MAX(D6:D7)		
9						
10	2	Chance	No oil (3)	=B16	0.7	=B10*E10
11	Drill at site 1		Oil	=C1*C2-C3	=1-E10	=B11*E11
12			Expected value			=SUM(F10:F11)
13						
14	3	Decision	Drill (4)	=F20		
15	No oil at site 1		Do not drill	=C3		
16			Maximum EV	=MAX(D14:D15)		
17						
18	4	Chance	No oil	=-2*C3	0.9	=D18*E18
19	Drill at site 2		Oil	=C1*C2-2*C3	=1-E18	=D19*E19
20			Expected value			=SUM(F18:F19)

Hiding the formulae, the calculated values will be:

	A	B	C	D	E	F
1		Oil reserves (m gallons)	16			
2		Profit per gallon (£)	1.00			
3		Cost of test drill (£m)	2.00			
4						
5	Node	Node Type	Branch (next node)	Value (£m)	Probability	Prob. X Value
6	1	Decision	Drill (2)	2.80		
7			Do not drill	0.00		
8			Maximum EV	2.80		
9						
10	2	Chance	No oil (3)	-2.00	0.70	-1.40
11	Drill at site 1		Oil	14.00	0.30	4.20
12			Expected value			2.80
13						
14	3	Decision	Drill (4)	-2.40		
15	No oil at site 1		Do not drill	-2.00		
16			Maximum EV	-2.00		
17						
18	4	Chance	No oil	-4.00	0.90	-3.60
19	Drill at site 2		Oil	12.00	0.10	1.20
20			Expected value			-2.40

A well-structured spreadsheet model is ideal for *sensitivity analysis*. Always try to be as flexible as possible. That means that cost or payoff data should normally only be typed in once, so that for ‘what-if’ analyses (such as “what if the cost of drilling were raised to £3 million?”) you only need to alter a single number in a single cell.

For example, the question “Would the drill/don’t drill decisions change if the profit per gallon rose to £1.50?” can be answered by just changing the profit figure in the spreadsheet (in cell C2) from £1 to £1.50. The spreadsheet recalculates the

expected values and makes fresh decisions, as follows:

	A	B	C	D	E	F
1		Oil reserves (m gallons)	16			
2		Profit per gallon (£)	1.50			
3		Cost of test drill (£m)	2.00			
4						
5	Node	Node Type	Branch (next node)	Value (£m)	Probability	Prob. X Value
6	1	Decision	Drill (2)	5.48		
7			Do not drill	0.00		
8			Maximum EV	5.48		
9						
10	2	Chance	No oil (3)	-1.60	0.70	-1.12
11	Drill at site 1		Oil	22.00	0.30	6.60
12			Expected value			5.48
13						
14	3	Decision	Drill (4)	-1.60		
15	No oil at site 1		Do not drill	-2.00		
16			Maximum EV	-1.60		
17						
18	4	Chance	No oil	-4.00	0.90	-3.60
19	Drill at site 2		Oil	20.00	0.10	2.00
20			Expected value			-1.60

More complex questions can be answered on the spreadsheet, either by trial and error or by using *Excel’s Goal Seek* tool. For example, we might wish to know the profit figure at which the decision maker is indifferent between drilling and not drilling at first site. This figure can be found algebraically, but *Goal Seek* can find it by iteratively changing the value in cell C2 until the difference between cells D6 and D7 is zero, i.e., until the expected payoff from drilling is identical to that from not drilling:

	A	B	C	D	E	F
1		Oil reserves (m gallons)	16			
2		Profit per gallon (£)	0.417			
3		Cost of test drill (£m)	2.00			
4						
5	Node	Node Type	Branch (next node)	Value (£m)	Probability	Prob. X Value
6	1	Decision	Drill (2)	0.00		
7			Do not drill	0.00		
8			Maximum EV	0.00		
9						
10	2	Chance	No oil (3)	-2.00	0.70	-1.40
11	Drill at site 1		Oil	4.67	0.30	1.40
12			Expected value			0.00
13						
14	3	Decision	Drill (4)	-3.33		
15	No oil at site 1		Do not drill	-2.00		
16			Maximum EV	-2.00		
17						
18	4	Chance	No oil	-4.00	0.90	-3.60
19	Drill at site 2		Oil	2.67	0.10	0.27
20			Expected value			-3.33

2.2 Shift Scheduling Example

Excel’s Solver tool can be used for optimisation to, for example, minimise the number of staff required on each shift pattern to provide cover on Monday-Friday at a service centre. Employees work four 10-hour days per week in any of the following shift patterns:

- Monday, Wednesday, Thursday, Friday

2. Monday, Tuesday, Thursday, Friday
3. Monday, Tuesday, Wednesday, Friday

The following linear programme (LP) determines the minimum number of employees needed to have at least 10 on duty on Mondays, 9 on Fridays and 7 from Tuesday through to Thursday.

Decision variables:

x_i = [integer] number of employees working pattern i for $i = 1, 2, 3$.

minimise $x_1 + x_2 + x_3$
[total staff required]

such that

- $$\begin{aligned}
 x_1 + x_2 + x_3 &\geq 10 && \text{[cover Monday]} \\
 x_2 + x_3 &\geq 7 && \text{[Tuesday]} \\
 x_1 + x_3 &\geq 7 && \text{[Wednesday]} \\
 x_1 + x_2 &\geq 7 && \text{[Thursday]} \\
 x_1 + x_2 + x_3 &\geq 9 && \text{[Friday]}
 \end{aligned}$$

and $x_1, x_2, x_3 \geq 0$ and integer

The Excel model has a very transparent layout, clearly showing the matrix of the underlying integer linear programme:

	A	B	C	D	E	F	G	H
1	Staff Scheduling Problem							
2		Shift pattern						
3		x1	x2	x3				
4	Number of Staff required	4	3	4				
5								
6	Objective Function (min)	1	1	1	11	(Total staff required)		
7								
8					Provided	Required	Surplus	
9	Cover Monday	1	1	1	11	>=	10	1
10	Cover Tuesday		1	1	7	>=	7	0
11	Cover Wednesday	1	1	1	8	>=	7	1
12	Cover Thursday	1	1		7	>=	7	0
13	Cover Friday	1	1	1	11	>=	9	2

Implementing Excel's Solver tool, shows that four staff are needed in shift pattern 1, three in pattern 2, and four in pattern 4, totalling a minimum number of eleven staff.

2.3 Example: Supply Chain.

Spreadsheets can also hide underlying formulas to give very concise and intuitive layouts, as the following supply chain transshipment example shows:

	A	B	C	D	E	F	G	H	
1	Transshipment Problem - Concise Layout								
2	Infinite Cost = 999								
3	Data								
4	Unit Costs	From Factory							
5	To	Plant	Ports	Depot D	Depot W	Depot M	Depot S	Requirements	
6	Depot D	0.5	999						
7	Depot W	0.5	0.3						
8	Depot M	1.0	0.5						
9	Depot S	0.2	0.2						
10	Customer 1	1.0	2.0	999	1.0	999	999	100	
11	Customer 2	1.5	999	0.5	0.5	2.0	0.2	80	
12	Customer 3	2.0	999	1.5	1.0	999	1.5	70	
13	Customer 4	999	999	999	0.5	0.5	0.5	120	
14	Customer 5	1.0	999	1.0	999	1.5	1.5	40	
15	Max Supply	300	400	140	100	200	80		
16									
17	Solution	From Factory							
18	Send	Plant	Ports	Depot D	Depot W	Depot M	Depot S	Total received	
19	To								
20	Depot D	0	0					0	
21	Depot W	0	100					100	
22	Depot M	0	90					90	
23	Depot S	80	0					80	
24	Customer 1	100	0	0	0	0	0	100	
25	Customer 2	0	0	0	0	0	80	80	
26	Customer 3	0	0	0	70	0	0	70	
27	Customer 4	0	0	0	30	90	0	120	
28	Customer 5	40	0	0	0	0	0	40	
29	Total sent	220	190	0	100	90	80		
30									
31	Total Cost =	£ 377							

2.4 Use of VBA

Visual Basic for Applications (VBA) is the programming language embedded in Microsoft Office. It can be used very simply, for example, to allow a user to solve an optimisation problem by pressing a **Solve LP** button as in the following spreadsheet model:

	A	B	C	D	E	F
1	Solve LP	Fuel	Solvent			
2		x1	x2			
3	Production Quantities	25	20	Values of Decision Variables		
4						
5	Profit margins (max)	£ 40	£ 30	£ 1,600		
6						
7				Requirements	Availability	
8	Material 1	0.4	0.5	20	<=	20
9	Material 2	0.6	0.3	21	<=	21
10	Material 3	0	0.2	4	<=	5

VBA code is edited within Excel. For example, the VBA code within the **Solve LP** button is:

```

on.xls - [Module1 (Code)]
Ln 7, Col 1
Run Tools Add-Ins Window Help

[General]

Option Explicit
Sub SolveLP()
    Range("Decision_Variable_Values").C
    SolverSolve UserFinish:=True
    SolverFinish KeepFinal:=1, ReportAr:
End Sub

```

2.5 Limitation of spreadsheets for OR

The examples above give an idea of the limitations of spreadsheets 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. Calculation time is usually (much) slower than in specialist software and OR functionality is more limited. For example, Excel Solver can only handle relatively small optimisation models whose coefficient matrix has already been generated.

Moreover, spreadsheets are notoriously prone to errors [8,9] which are frequently not obvious and may not be obvious, creating a dangerous over-confidence in calculation results. Even if detected, errors hidden in spreadsheet formulas can be difficult to find.

VBA does permit the automation of behind-the-worksheet processing and allows intermediate calculations to be hidden off-sheet enabling a clearer spreadsheet. VBA code can replace long formulas or many cells, resulting in fewer errors. There is a VBA programme development and debugging within Excel, so that further development software is not needed. In addition, VBA can automatically integrate Excel with Word & Powerpoint.

However, VBA code is often neither obvious to understand nor transparent. The learning-curve is steep, slow, and easily forgotten. Furthermore, certain mundane tasks are difficult, for example, it is complicated to read a text file word-by-word rather than line-by-line as in VBA.

As a result it is often cumbersome, limiting 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 for optimisation is to use a modelling language, as explained in the next section.

3. MODELLING LANGUAGES

A developer of an optimisation modelling language and system [10], argues that “*a spreadsheet approach works well when:*

- *you do not need to specify a large number of relationships,*
- *there are only a few procedures to be written,*
- *the size of your data sets will remain stable,*
- *the need to add or remove dimensions is limited, and*
- *you will carry out all the maintenance activities yourself.”*

These conditions are indeed met for many straightforward end-user modelling application, but only in a minority of cases when considering more complex systems. For such systems, a modelling language can be useful, because:

- specifying a large number of variables and constraints is simple using indices,
- modifying the size of any index set is natural, as there is a complete separation between model and data,
- adding or removing dimensions takes place in the language, and
- adding and managing both internal and external procedures is straightforward.

There are several such languages and systems, including AIMMS [10], GAMS [11], XPress-MP [12], OPL [13] and AMPL [14]. This paper will focus on AMPL (A Mathematical Programming Language), used for both teaching [15] and research [16,17,18] by the author at the University of the West of England.

3.1 The AMPL modeling language

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

Plant / Product	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	

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 modeling languages), data is separated from the model whereas they are missed 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 is declared, 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 (named, for example, product.mod). The data is supplied separately in another file (named, 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 3 2;

```

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 [19], 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 Cplex version 10.0.1 was used to solve the LP/MIP
- that an optimal solution was obtained with objective value \$36,000
- that 0 dual simplex iterations were used in finding the solution
- the solution results outputted using the `display` command.

3.2 Increasing the instance size

Let us now 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;
```

giving 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

3.3 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
- that Cplex obtained an optimal solution with objective value \$46,800 [less profitable than the previous non-integer solution].

3.4 A more complex example (Critical Path in a Project)

Consider a project with 26 activities (A-Z). To calculate the Earliest Start Times (ESTs) of the activities and thus the project's shortest possible duration, a minimising LP is solved. To calculate the Latest Start Times (LSTs) of the activities, and thus identify the project's critical path, a maximizing LP is solved, using as input the project duration output by the first LP. In AMPL, this is achieved as follows.

The model file `EST-LST.mod` is:

```
var ActivityStartTime {i in
Activities} >= 0;
```

```
# (earliest/latest) start time of
activity i
```

```
minimize Minimize_Start_Times:
sum {i in Activities}
ActivityStartTime[i];
```

```
maximize Maximize_Start_Times:
sum {i in Activities}
ActivityStartTime[i];
```

```
subject to
Activity_Precedence_Constraints
{i in Activities,
j in Activities : j in P[i]}:
ActivityStartTime[i] >=
ActivityStartTime[j] +
d[j];
```

```
subject to Fix_Project_Duration:
ActivityStartTime["End"]
= EST["End"];
```

Observe that two objective functions have been declared and specified. Note also that the set `Activities` has not (apparently) been declared, nor has the parameter `P`. In fact, both are declared in the run file:

```
option solver cplex;
option show_stats 1;
option cplex_options 'timing=1
mipdisplay=1';
```

```
# Set of project activities
set Activities;
# Duration of each activity
param d {Activities} >= 0 integer;
# Predecessor activities of each
activity
set P {Activities} within
Activities;
# Earliest and Latest Start Times
of each activity
param EST {Activities} >= 0;
param LST {Activities} >= 0;
```

```
model EST-LST.mod;
data Project.dat;
```

```
problem Find_ESTs:
ActivityStartTime,
Minimize_Start_Times,
Activity_Precedence_Constraints;
```

```
problem Find_LSTs:
ActivityStartTime,
Maximize_Start_Times,
Activity_Precedence_Constraints,
Fix_Project_Duration;
```

```

param NumberOfActivities integer >
0;

# Calculate the number of
activities
let NumberOfActivities := 0;
for {i in Activities : i <> "End" }
{
    let NumberOfActivities
    := NumberOfActivities + 1;
}

problem Find_ESTs;
solve;

let {i in Activities} EST[i]
:= ActivityStartTime[i];

printf "\nProject duration = %d
days\n", EST["End"];

printf "\nEarliest Activity Start
Times ";
display EST;

problem Find_LSTs;
solve;

let {i in Activities} LST[i] :=
ActivityStartTime[i];
printf "\nLatest Activity Start
Times ";
display LST;

```

This run file illustrates several powerful features of AMPL. Note the

- `show_stats` option with value 1;
- `cplex timing` and `display` options, both with value 1;
- declaration and definition of two distinct problems (with names `Find_ESTs` and `Find_LSTs`) by specifying the objective function and constraints associated with each problem;
- procedure to calculate the number of activities;
- the activation, solving and output of the solution of problem `Find_ESTs`;
- the activation and solving of problem `Find_LST`, using the value of `EST["End"]` output by the solution of problem `Find_ESTs`;
- the output of the solution of problem `Find_LSTs`.

The data file `Project.dat` is:

```

data;

set Activities :=
A
B
C
...
Z
End;

param d :=
A 10
B 1
C 1
...
Z 1;

set P[A] := ;
set P[B] := A;
set P[C] := ;
set P[D] := ;
...
set P[End] := B J Z;

```

The solution output is:

27 variables, all linear
26 constraints, all linear; 52
nonzeros
1 linear objective; 27 nonzeros.

```

CPLEX 8.0.0: timing=1
vartol=0
mipdisplay=1
timelimit=60

```

```

Times (seconds):
Input = 2.934
Solve = 0.491
Output = 0.01
CPLEX 8.0.0: optimal solution;
objective 174
7 dual simplex iterations (0 in
phase I)

```

The project duration is 16 days

```

Earliest Activity Start Times EST
[*] :=
A 0   D 1   F 3   I 6   L 1
O 4   R 7   U 10  X 13  B 10
E 2   G 4   J 7   M 2   P 5
S 8   V 11  Y 14
C 0   End 16 H 5   K 0   N 3
Q 6   T 9   W 12  Z 15;

```

```

Presolve eliminates 19 constraints
and 17 variables.
Adjusted problem:
10 variables, all linear

```

```

8 constraints, all linear; 16
nonzeros
1 linear objective; 10 nonzeros.

CPLEX 8.0.0: timing=1
varsel=0
mipdisplay=1
timelimit=60

Times (seconds):
Input = 0.01
Solve = 0
Output = 0
CPLEX 8.0.0: optimal solution;
objective 248
2 simplex iterations (0 in phase I)

Latest Activity Start Times LST [*]
:=
A 5 D 9 F 11 I 14 L 1
O 4 R 7 U 10 X 13 B 15
E 10 G 12 J 15 M 2 P 5
S 8 V 11 Y 14
C 8 End 16 H 13 K 0 N 3
Q 6 T 9 W 12 Z 15;

```

The critical path can now be identified with a few more lines of AMPL code (not shown here).

3.5 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 costs a lot of money [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 [20].

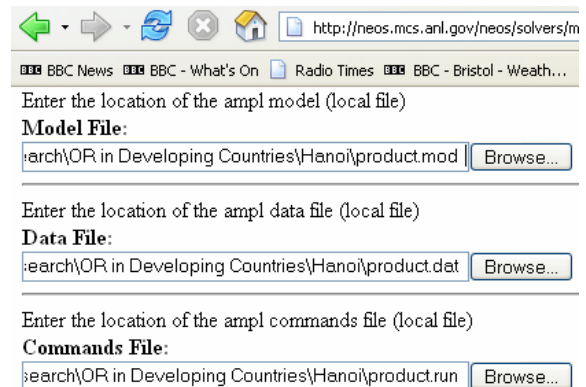
3.4 GNU Linear Programming Kit

The GNU Linear Programming Kit (GLPK) [21] is

- free (and legal);
- contains the GNU MathProg language, a subset of AMPL;
- provides the GLPSol solver, but which is much slower than Cplex;
- is a set of routines organized into a callable library within the C programming language;
- but MathProg and GLPSol can be used alone without using C;

3.5 The NEOS Server

The NEOS server [22,23,24,25] is open and free to the public for the optimization of large models within certain time or solver-dependent iteration limits. A user submits online a model specified in one of the input formats accepted by a solver (such as AMPL), a data set and a run file.



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 all take input from AMPL. One which does is Minto [26], as used above.

3.6 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 solver CLP [27] through Open Solver Interface (OSI) of the Computational Infrastructure for Operations Research (COIN-OR) project [28], 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.

4. CONCLUSION

This paper has shown that sophisticated decision-making technology is available to persons and organisations worldwide without having to purchase expensive

optimisation software. The use of such technology often involves the judicious combination of tools from disparate sources, but usually not in the seamless manner that a lay-user desires. It requires competence, confidence and patience in the use of the internet, command-line programming tools, text file editing, and spreadsheets, qualities possessed by the typical quantitatively-minded and technically competent student or organisational employee.

5. ACKNOWLEDGEMENTS

I would like to thank the Hanoi University of Technology for their kind invitation to attend and present this paper at the 20th Scientific International Conference, Hanoi, Vietnam, 12-13 October 2006. Much of the the material and examples have come from course notes written by myself and other colleagues in the School of Mathematical Sciences at the University of the West of England.

6. REFERENCES

1. **INFORMS**, Operations Research: The Science of Better. Website: www.scienceofbetter.org
2. **M Taube-Netto**, Integrated Planning for Poultry Production at Sadia, *Interfaces*, 26 (1), 1996, pp. 38-53.
3. **A. C. Weintraub, R. L. C. Church, A. T. C. Murray, M. C. Guignard**, Forest management models and combinatorial algorithms: analysis of state of the art, *Annals of Operations Research*, 96, 2000, pp 271-285.
4. **A. Martin**, An integrated introduction to spreadsheet and programming skills for operational research students, *Journal of the Operational Research Society* 51, 2000, pp 1399-1408
5. **Google spreadsheets**, online at spreadsheets.google.com
6. **S. G. Powell**, End-user modeling, *OR/MS Today*, 24 (4), 1997.
7. **T. A. Grossman**, End-user modeling, *OR/MS Today*, 24 (5), 1997, downloadable at

www.lionhrtpub.com/orms/orms-10-97/IiE.html

8. **J. Caulkins**, Do spreadsheet errors harm executive decision-making? New evidence from interviews. INFORMS Denver, 2004.

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

10. **AIMMS**, Paragon Decision Technology, B.V., P.O. Box 3277, 2001 DG Haarlem, The Netherlands. Website: www.aimms.com

11. **GAMS** - General Algebraic Modeling System, GAMS Development Corporation 1217 Potomac Street, NW Washington, DC 20007, USA, Website: www.gams.com

12. **XPress-MP**, Dash Optimization Ltd, Blisworth House, Church Lane, Blisworth, Northamptonshire, NN7 3BX, England. Website: www.dashoptimization.com

13. **OPL** - Optimization Programming Language, ILOG SA, 9, rue de Verdun, BP 85, 94253 Gentilly Cedex, France. Website: www.ilog.com

14. **R. Fourer, D. M. Gay, & B. W. Kernighan**; AMPL: A Modeling Language for Mathematical Programming, 2nd edition, Duxbury Press / Brooks/Cole Publishing Company, 2002, ISBN 0-534-38809-4. Website: www.ampl.com

15. **A. R. Clark**, Combinatorial Optimisation and Heuristics, postgraduate module UFQEF8-15-M, Faculty of Computing, Engineering and Mathematical Sciences, University of the West of England, Bristol, BS16 1QY, England. Webpage: www.cems.uwe.ac.uk/~arclark/coh

16. **A. R. Clark**, Optimization Approximations for Capacity Constrained Material Requirements Planning Problems, *International Journal of Production Economics* 84, 2003, pp 115-131.

17. **A. R. Clark**, Hybrid Heuristics for Planning Lot Sizes and Setups, *Computers and Industrial Engineering* 45, 2003, pp 545-562.

18. **A. R. Clark**, Rolling horizon heuristics for Production and Setup Planning with Backlogs and Error-prone Demand Forecasts",

Production Planning and Control 16, 2005, pp 81-97.

19. **Cplex Mathematical Programming Optimizer**, ILOG SA, 9, rue de Verdun, BP 85, 94253 Gentilly Cedex, France. Website: www.cplex.com

20. **J. T. Linderoth and T. K. Ralphs**, Noncommercial Software for Mixed-Integer Linear Programming, in *Integer Programming: Theory and Practice*, John Karlof (ed.), CRC Press Operations Research Series, 2005, pp 253-303, available online at webpage: coral.ie.lehigh.edu/pubs/files/jt13_noncomm.pdf.

21. **GNU Linear Programming Kit (GLPK)**, freely downloadable from www.gnu.org/software/glpk

22. **NEOS Server for Optimization**, online at www-neos.mcs.anl.gov

23. **J. Czyzyk, M. Mesnier, and J. Moré**, The NEOS Server, *IEEE Journal on Computational Science and Engineering* 5, 1998 pp 68-75.

24. **W. Gropp and J. Moré**, Optimization Environments and the NEOS Server, *Approximation Theory and Optimization*, (M. D. Buhmann and A. Iserles, eds), pp 167-182, Cambridge University Press, 1997.

25. **E. Dolan**, The NEOS Server 4.0 Administrative Guide, Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, May 2001.

26. **MINTO** (Mixed INTeger Optimizer), Website: coral.ie.lehigh.edu/minto

27. **CLP Linear Program Solver**, Website: www.coin-or.org/Clp

28. **COIN-OR (Computational Infrastructure for Operations Research)**, Website: www.coin-or.org

-o0o-