Amsterdam Optimization Modeling Group LLC

Modeling with Excel+OML, a practical guide

This document describes the use of Microsoft's OML language to specify Mathematical Programming Models. The emphasis is on modeling rather than programming. A number of actual annotated models is presented to illustrate how this new modeling system can be used to implement and solve practical problems.

The models in this paper are based on MS Solver Foundation 1.x.

Erwin
10/16/2009

1 CONTENTS

2 INTRODUCTION

Microsoft has a long history with offering optimization tools. Inside Excel is **Solver** which can solve linear programming problems, mixed-integer programming problems and non-linear problems (Fylstra, Lasdon, Watson, & Warren, 1998). Arguably this makes Microsoft the most successful vendor of optimization software with over 500 million copies distributed[1].

The Excel solver is using cells and cell-references to formulate and implement optimization models. This has an obvious advantage: Excel users are directly familiar with this structure and can build optimization models without a steep learning curve. The direct implementation of a model in Excel has of course numerous benefits such as availability of data manipulation tools and report writing facilities including the availability of numerous built-in functions, dynamic graphs, and pivot tables. There are also some serious disadvantages: spreadsheet modeling is error prone[2], the model lacks structure and dimensionality and we are missing out on a short, compact representation of the model that allows us to think and talk about it and to share it with colleagues.

Modeling is difficult. Practical optimization models are often messy with ad-hoc and unstructured “business rules” (to use a buzzword). This means that any help in adding structure to the model is very welcome. Opposed to usual computer programming where we can often break down a complicated part into smaller more manageable pieces (Wirth, 1971) we deal essentially with systems of simultaneous equations. In this situation stepwise refinement into smaller entities is often not a viable strategy: we need to look at the model as a whole. A high-level modeling language can help here: it can provide a compact representation of the model that allows the modeler to view and comprehend a complete model. This in turn will allow the modeler to adapt and maintain the model with much more ease than otherwise possible. Compared to using a modeling language, the use of a solver API (Application Programming Interface) is really a step backward: it will create a more cluttered, wieldy expression of a model that makes maintenance and experimentation more difficult, time-consuming and expensive. For very structured or very small models this may be not prohibitive, but for large, complex models a good modeling language is an invaluable tool that can make a modeler more efficient by orders of magnitude.

The new Microsoft Solver Foundation product is what we focus on in this document. MSF consists of a number of modules:

- Solvers (LP, MIP, CSP)
- OML: an equation based modeling language
- API’s: programming interfaces allowing programmers to talk to Solver Foundation services
- Solver plug-in capabilities: external solvers can be hooked up. With version 1.1 the state-of-the-art Gurobi MIP solver has become the default MIP solver. It is accessed through this mechanism.
- An Excel based framework to develop and solve OML models

We will concentrate on the modeling language OML and the Excel application framework.

2.1 MODELING LANGUAGE VS. API

A model can be built using a modeling language or using a traditional programming language such as C. In the latter case one can use a solver API (Application Programming Interface) to assemble the model. We see that many beginners in modeling are attracted to using the API, especially if they have a background and experience in computer programming. In my opinion this API-appetite is often unwarranted: if the model is not either very small or large but very structured, expressing the model in a specialized modeling language is by far preferable in terms of productivity, performance, quality and maintainability of the model.

Developing a model in a modeling language is often much more efficient. First of all, a model expressed in a modeling language is much more compact. The same model in a programming language will require many more lines of code. Further we often see modelers struggling with low level issues like memory allocation, pointer problems and obscure linker messages that simply do not occur when using a modeling language. The gain in productivity can be used spend more time to improve the model formulation.

Large, difficult models require many revisions and experiments to achieve best performance (speed and reliability). Different formulations can lead to large differences in performance, so it is beneficial if it is easy to try out different formulations quickly. Here a modeling language shines compared to a traditional programming language.

The most well-known modeling languages are GAMS and AMPL. They are both fairly complex systems, and there is a learning curve before you are comfortable with these languages. But once you mastered them, you can build large, complex, maintainable models in a controlled fashion. OML is a much simpler language. Much of the complexity (such as any data manipulation) is moved away from the modeling language to the environment where OML is called from. This can be a C# program, or in the case of this paper Excel. This approach comes with some advantages (a simpler, cleaner modeling language) and disadvantages (more complex and precise data preparation is needed before we can pass data on to the OML model to form a complete model instance). In this paper we will explore some of these issues.

We will focus on OML as used in the Excel plug-in. The tight integration between Excel and OML gives a rich but unstructured environment for data handling and reporting, a small, limited modeling language, a build-in scripting language (VBA) and enough widgets such as buttons to create mini-applications.

2.2 A TRANSPORTATION MODEL

The transportation model is among the simplest Linear Programming models we can present.

We want to minimize shipping cost while obeying demand and supply restrictions. The mathematical model can be stated as:

$$\begin{gathered}\min \textstyle\sum_{i,j} c_{i,j}\ x_{i,j} \\ \textstyle\sum_i x_{i,j} \ \geq d_j \ \ \forall j \\ \textstyle\sum_j x_{i,j} \ \leq s_i \ \ \forall i \\ x_{i,j} \ \geq 0\end{gathered}$$

Here *x* is the decision variable and *c*, *d*, and *s* are parameters. The difference between a parameter and a decision variable is an important one. A parameter is a constant during the solution of the model: it will not be changed by the solver. A variable will be changed by the solver: hopefully it will return the best possible values for the decision variables.

In this section we will compare the OML representation of this model to two alternatives: a GAMS formulation and an implementation in Excel Solver.

2.2.1 A GAMS REPRESENTATION

The first model in the Model Library from GAMS is a simple example of this problem, based on the famous text book (Dantzig, 1963)[3]. The complete model looks like:

```
$Title  A Transportation Problem (TRNSPORT,SEQ=1)
$Ontext

This problem finds a least cost shipping schedule that meets
requirements at markets and supplies at factories.


Dantzig, G B, Chapter 3.3. In Linear Programming and Extensions.
Princeton University Press, Princeton, New Jersey, 1963.

This formulation is described in detail in:
Rosenthal, R E, Chapter 2: A GAMS Tutorial. In GAMS: A User's Guide.
The Scientific Press, Redwood City, California, 1988.

The line numbers will not match those in the book because of these
comments.

$Offtext

  Sets
       i   'canning plants'   / seattle, san-diego /
       j   'markets'          / new-york, chicago, topeka / ;

  Parameters

       a(i)  'capacity of plant i in cases'
         /    seattle     350
              san-diego   600  /

       b(j)  'demand at market j in cases'
         /    new-york    325
              chicago     300
              topeka      275  / ;

  Table d(i,j)  'distance in thousands of miles'
                    new-york       chicago      topeka
      seattle          2.5           1.7          1.8
      san-diego        2.5           1.8          1.4  ;

  Scalar f  'freight in dollars per case per thousand miles'  /90/ ;
```

```
Parameter c(i,j)  'transport cost in thousands of dollars per case' ;

          c(i,j) = f * d(i,j) / 1000 ;

Variables
     x(i,j)  'shipment quantities in cases'
     z       'total transportation costs in thousands of dollars' ;

Positive Variable x ;

Equations
     cost        'define objective function'
     supply(i)   'observe supply limit at plant i'
     demand(j)   'satisfy demand at market j' ;

cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;

demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;

Solve transport using lp minimizing z ;

Display x.l, x.m ;
```

The sets indicate collections of strings that we use for indexing. GAMS uses string as vehicle for indexing vectors and matrices. This has some advantages: it makes the model easier to read (plant 'Seattle' is more descriptive than plant 1), and it makes it unattractive to use index arithmetic in cases where this may not be needed. The latter is also a negative as the obvious disadvantage is that it makes index arithmetic more complicated where we can legitimately use it.

Parameter, scalar and table statements are used to specify parameters. Parameters can be changed inside the GAMS model (using assignment statements) but not by the solver. During the SOLVE statements parameters are constants. GAMS allows for convenient data entry: only the nonzero elements need to be provided in parameter and table statements. Data manipulation is done by assignment statements, like `c(i,j) = f * d(i,j) / 1000` which can be interpreted as an implicit loop.

The optimization model itself starts with variable and equation declarations. By default variables are free, i.e. they are allowed to assume positive and negative values. With `Positive Variable x` we impose a lower bound of zero. The equations are declared with a somewhat peculiar syntax. Equality is denoted by =e= while =l= and =g= are less-than-or-equal and greater-than-or-equal constraints. Note that each constraint is actually a block of constraints. E.g. constraint `demand(j)` implements three constraints because set j has three elements.

It is important to understand the difference between assignment statements and equations. Assignments are executed by GAMS itself in order as they appear, while equations are passed on the solver and must hold simultaneously. In programming language parlor we say that data manipulation (i.e. assignments) is procedural while model equations are declarative.

Finally we have a model and solve statement and the results are displayed. GAMS has the notion of an objective variable opposed to an objective function. In practice this is not a problem: just place your objective in an equality constraint, and optimize the corresponding variable. Note that `x.l, x.m` indicates we want to see the optimal level values and the optimal marginal values (or reduced cost) of x.

The results of a GAMS job are written to a listing file. The listing file will contain:

- A source listing of the model. This can be useful to find the location of syntax errors or run-time errors.
- A listing of individual rows and columns generated by the model, i.e. the expanded model. This is useful for debugging.
- A section with messages from the solver. Hopefully it will say OPTIMAL.
- The solution: rows and columns. Both level values and marginals are printed. Marginals are reduced costs for variables and duals for equations.
- The output of display statements.

GAMS has built-in facilities for report-writing: we can use data-manipulation on solution vectors, and display the final results.

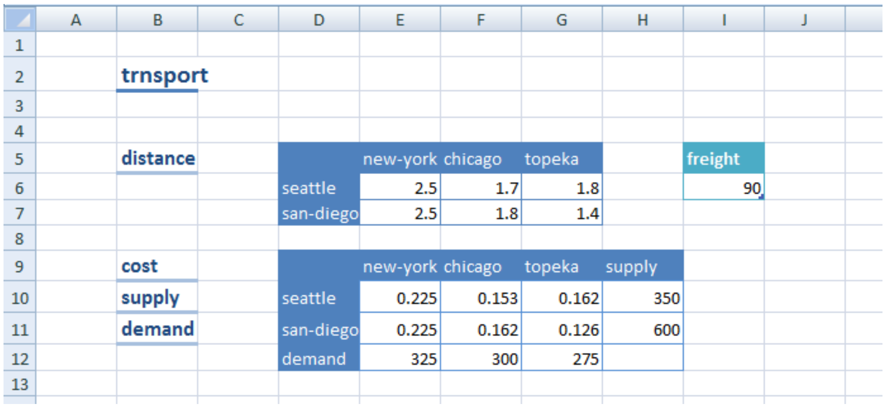
For large models, GAMS has a number of facilities:

- All data structures are sparse: no storage for zero's
- $ conditions allow implementing 'such that' operations on sets
- Abort statement for error checking
- Loop statement to handle multiple solve statements, e.g. to implement heuristics
- GAMS comes with an IDE (under Windows)

The integration with Excel is limited. There is an external program (gdxxrw.exe) that allows for exchanging data between GAMS and Excel, but to use this from an active Excel spreadsheet is difficult. It requires a fairly large amount of VBA code to run GAMS from Excel. See http://www.amsterdamoptimization.com/packaging.html.

2.2.2 AN EXCEL SOLVER APPROACH

The traditional way to model this problem in Excel is to use Solver. First we setup the data. This includes unit cost coefficients *c*, supply capacity *s* and demand data *d*. In our case we calculate the cost coefficients from unit transportation cost and a distance table.



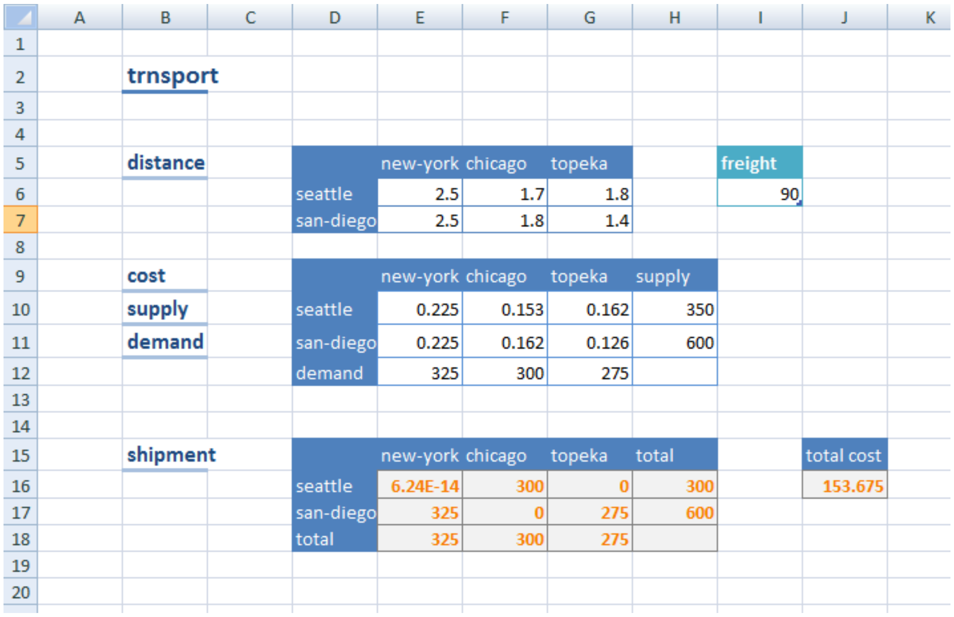
	A	B	C	D	E	F	G	H	I	J
1										
2		trnsport								
3										
4										
5		distance			new-york	chicago	topeka		freight	
6				seattle	2.5	1.7	1.8		90	
7				san-diego	2.5	1.8	1.4			
8										
9		cost			new-york	chicago	topeka	supply		
10		supply		seattle	0.225	0.153	0.162	350		
11		demand		san-diego	0.225	0.162	0.126	600		
12				demand	325	300	275			
13										

Figure 1. Data for the TRNSPORT model

The cost data are calculated by the formula:

$$c_{i,j} = \frac{f \cdot d_{i,j}}{1000}$$

E.g. cell E10 has formula =`E6*Freight/1000`. Now we setup a table where the shipments will be placed.



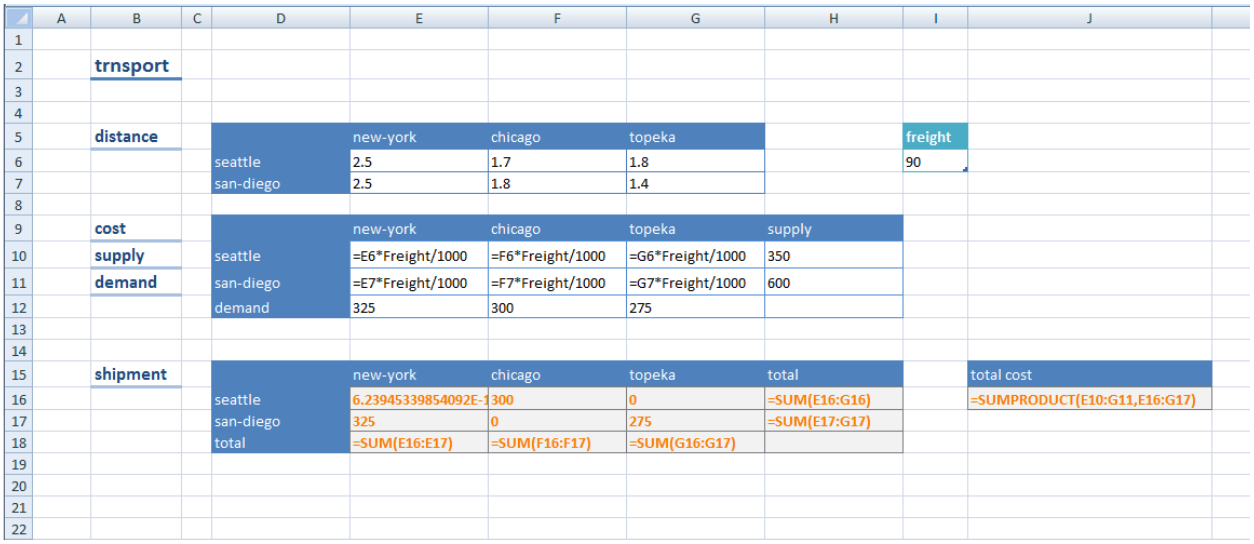
	A	B	C	D	E	F	G	H	I	J	K
1											
2		trnsport									
3											
4											
5		distance			new-york	chicago	topeka		freight		
6				seattle	2.5	1.7	1.8		90		
7				san-diego	2.5	1.8	1.4				
8											
9		cost			new-york	chicago	topeka	supply			
10		supply		seattle	0.225	0.153	0.162	350			
11		demand		san-diego	0.225	0.162	0.126	600			
12				demand	325	300	275				
13											
14											
15		shipment			new-york	chicago	topeka	total		total cost	
16				seattle	6.24E-14	300	0	300		153.675	
17				san-diego	325	0	275	600			
18				total	325	300	275				
19											
20											

Figure 2. Derived data

The cells E16:G17 correspond to $x_{i,j}$. In addition we calculate the row and column sums. E.g. cell H16 has formula =`SUM(E16:G16)`. The total cost are calculated by: =`SUMPRODUCT(E10:G11,E16:G17)` which represents the expression

$$\sum_{i,j} c_{i,j} x_{i,j}$$

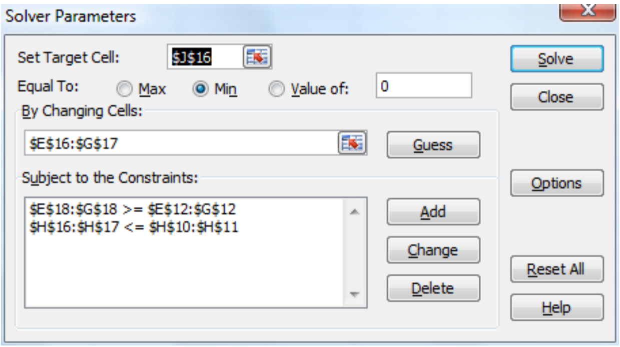
A complete view of the formulas is:



	A	B	C	D	E	F	G	H	I	J
1										
2		trnsport								
3										
4										
5		distance			new-york	chicago	topeka		freight	
6				seattle	2.5	1.7	1.8		90	
7				san-diego	2.5	1.8	1.4			
8										
9		cost			new-york	chicago	topeka	supply		
10		supply		seattle	=E6*Freight/1000	=F6*Freight/1000	=G6*Freight/1000	350		
11		demand		san-diego	=E7*Freight/1000	=F7*Freight/1000	=G7*Freight/1000	600		
12				demand	325	300	275			
13										
14										
15		shipment			new-york	chicago	topeka	total		total cost
16				seattle	6.23945339854092E-1	300	0	=SUM(E16:G16)		=SUMPRODUCT(E10:G11,E16:G17)
17				san-diego	325	0	275	=SUM(E17:G17)		
18				total	=SUM(E16:E17)	=SUM(F16:F17)	=SUM(G16:G17)			
19										
20										
21										
22										

Figure 3. Excel Solver model formulation

The model can now be specified in the solver



Solver Parameters

Set Target Cell: $J$16

Equal To: ○ Max ● Min ○ Value of: 0

By Changing Cells:

$E$16:$G$17

Subject to the Constraints:

$E$18:$G$18 >= $E$12:$G$12
$H$16:$H$17 <= $H$10:$H$11

Figure 4. Setup Solver

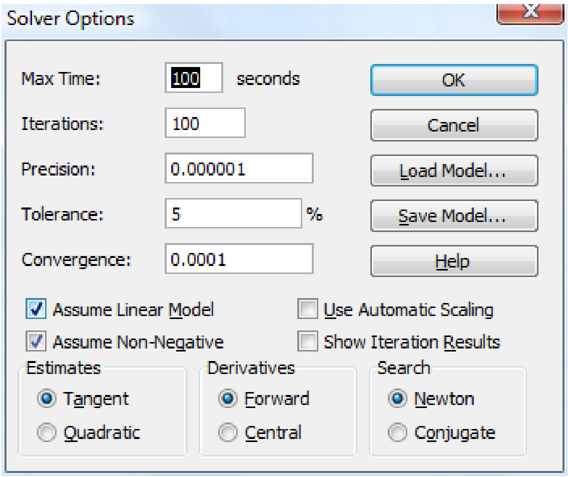
Here we specify:

- The objective is total cost at cell J16
- The variables $x_{i,j}$ are located in the table E16:G17
- The constraints are formed by calculating the sums $\sum_j x_{i,j}$ and $\sum_i x_{i,j}$ and comparing those quantities with the available supply and required demand.
- The conditions $x_{i,j} \geq 0$ are specified in the options dialog where we check the “assume non-negative” option.
- In addition we specify the model to be linear by checking “assume linear model”.

The disadvantage of this approach is visible here: it is difficult to recognize the model

$$\begin{gathered}\min \sum_{i,j} c_{i,j}\ x_{i,j} \\ \sum_i x_{i,j} \geq d_j \quad \forall j \\ \sum_j x_{i,j} \leq s_i \quad \forall i \\ x_{i,j} \geq 0\end{gathered}$$

in this spreadsheet. For a small or well-structured model this may not be a problem, but for more complex models, the lack of structure will be a major obstacle to build and maintain models efficiently. Indeed I have converted a number of Excel Solver models, and the majority of them contained errors such that I could not reproduce the results when using a GAMS model (in different words: the Excel solution was wrong).



Solver Options

Max Time: 100 seconds

Iterations: 100

Precision: 0.000001

Tolerance: 5 %

Convergence: 0.0001

☑ Assume Linear Model

☐ Use Automatic Scaling

☑ Assume Non-Negative

☐ Show Iteration Results

Estimates: ● Tangent ○ Quadratic

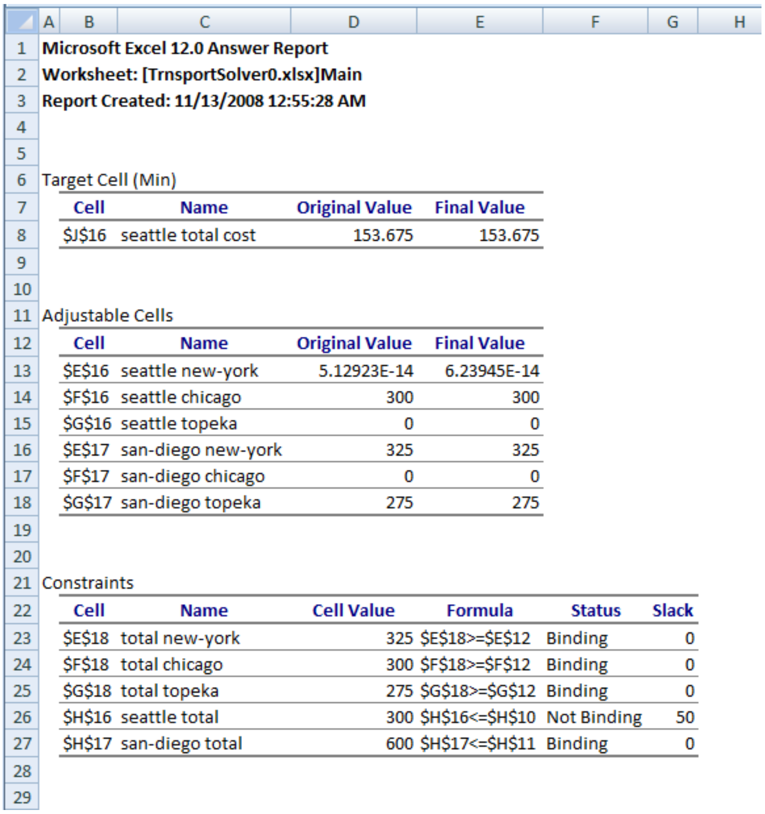
Derivatives: ● Forward ○ Central

Search: ● Newton ○ Conjugate

Figure 5. Excel Solver Options

The formulation can be improved by using named ranges.

The optimal values are updated in the spreadsheet. In addition the solver can generate a solution report as follows:



Microsoft Excel 12.0 Answer Report
Worksheet: [TrnsportSolver0.xlsx]Main
Report Created: 11/13/2008 12:55:28 AM

Target Cell (Min)

Cell	Name	Original Value	Final Value
$J$16	seattle total cost	153.675	153.675

Adjustable Cells

Cell	Name	Original Value	Final Value
$E$16	seattle new-york	5.12923E-14	6.23945E-14
$F$16	seattle chicago	300	300
$G$16	seattle topeka	0	0
$E$17	san-diego new-york	325	325
$F$17	san-diego chicago	0	0
$G$17	san-diego topeka	275	275

Constraints

Cell	Name	Cell Value	Formula	Status	Slack
$E$18	total new-york	325	$E$18>=$E$12	Binding	0
$F$18	total chicago	300	$F$18>=$F$12	Binding	0
$G$18	total topeka	275	$G$18>=$G$12	Binding	0
$H$16	seattle total	300	$H$16<=$H$10	Not Binding	50
$H$17	san-diego total	600	$H$17<=$H$11	Binding	0

Figure 6. Excel Solver Report

2.2.3 THE OML IMPLEMENTATION

The OML model to represent this mathematical model could look like:

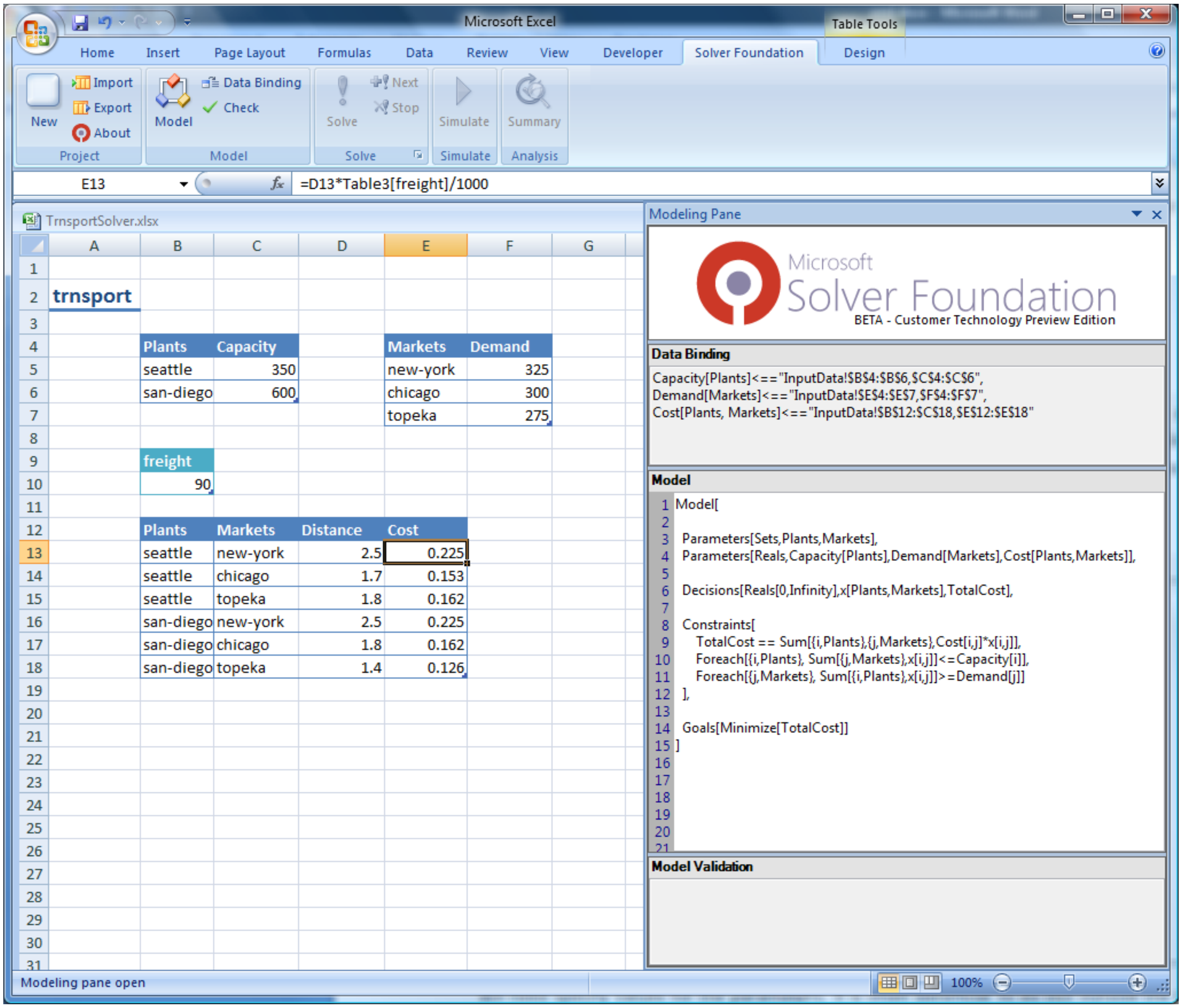
```
Model[

  Parameters[Sets,Plants,Markets],
  Parameters[Reals,Capacity[Plants],Demand[Markets],Cost[Plants,Markets]],

  Decisions[Reals[0,Infinity],x[Plants,Markets],TotalCost],

  Constraints[
      TotalCost == Sum[{i,Plants},{j,Markets},Cost[i,j]*x[i,j]],
      Foreach[{i,Plants}, Sum[{j,Markets},x[i,j]]<=Capacity[i]],
      Foreach[{j,Markets}, Sum[{i,Plants},x[i,j]]>=Demand[j]]
  ],

  Goals[Minimize[TotalCost]]
]
```

Figure 7. TRNSPORT model in OML

This model only has the declarative part of the corresponding GAMS model: declaration statements and equation definitions.

The sets are declared as part of parameter statement: they are parameter of a special type. The variables are called Decisions, and we can specify bounds on them in the declaration. By default OML assumes variables are free, just like GAMS (and unlike many other mathematical programming languages). Finally we specify the equations.

The data and the data manipulation need to be handled somewhere else. In this case we choose the spreadsheet interface. The screen-shot below shows how the data can be organized and entered in Excel:



E13 =D13*Table3[freight]/1000

trnsport

Plants	Capacity
seattle	350
san-diego	600

Markets	Demand
new-york	325
chicago	300
topeka	275

freight
90

Plants	Markets	Distance	Cost
seattle	new-york	2.5	0.225
seattle	chicago	1.7	0.153
seattle	topeka	1.8	0.162
san-diego	new-york	2.5	0.225
san-diego	chicago	1.8	0.162
san-diego	topeka	1.4	0.126

Data Binding

```
Capacity[Plants]<=="InputData!$B$4:$B$6,$C$4:$C$6",
Demand[Markets]<=="InputData!$E$4:$E$7,$F$4:$F$7",
Cost[Plants, Markets]<=="InputData!$B$12:$C$18,$E$12:$E$18"
```

Model

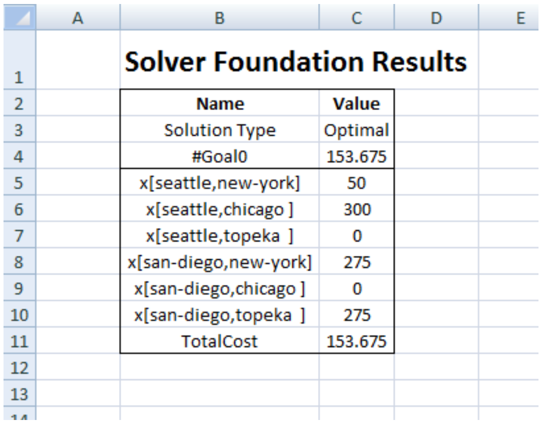
```
Model[

 Parameters[Sets,Plants,Markets],
 Parameters[Reals,Capacity[Plants],Demand[Markets],Cost[Plants,Markets]],

 Decisions[Reals[0,Infinity],x[Plants,Markets],TotalCost],

 Constraints[
    TotalCost == Sum[{i,Plants},{j,Markets},Cost[i,j]*x[i,j]],
    Foreach[{i,Plants}, Sum[{j,Markets},x[i,j]]<=Capacity[i]],
    Foreach[{j,Markets}, Sum[{i,Plants},x[i,j]]>=Demand[j]]
 ],

 Goals[Minimize[TotalCost]]
]
```

Model Validation

Figure 8. Data Binding for the TRNSPORT model

The data is organized in a table format in this example. The data binding is slightly different for table formatted data or block formatted data.

The results are written to a separate sheet, formatted by Solver Foundation.



Solver Foundation Results

Name	Value
Solution Type	Optimal
#Goal0	153.675
x[seattle,new-york]	50
x[seattle,chicago]	300
x[seattle,topeka]	0
x[san-diego,new-york]	275
x[san-diego,chicago]	0
x[san-diego,topeka]	275
TotalCost	153.675

Figure 9. Solver Foundation Solution Report

This modeling approach is a little bit between GAMS and the old Solver based structure: the model is specified in an equation-based modeling language OML, while all data manipulation and reporting is done in Excel.

2.2.4 MICROSOFT SOLVER FOUNDATION

The Solver Foundation comes standard with a number of solvers:

- LP solvers: there are several linear programming solvers: an interior point solver and a simplex based primal and dual solvers
- MIP solver: the Mixed Integer solver allows discrete variables. From version 1.1 the default MIP solver is Gurobi.
- QP solver: the interior point solver can also solve QP problems, i.e. models with a quadratic objective
- CSP solver: a solver for constraint programming problems is available

Compared to GAMS and AMPL we miss a general non-linear programming functionality – MSF contains an unconstrained NLP solver but this is not supported by OML. On the flip-side MSF offers a CSP solver. GAMS has no facilities to support CSP models, and in the case of AMPL some work has been done in this direction (Fourer, 1998), but to my knowledge this is not available in the official distribution. An example of a general purpose modeling language with support for constraint programming is OPL (van Hentenryck, 1999).

Now we have seen how the `trnsport` model can be implemented in three different ways, we will continue with the OML/Excel combination. We will explain the language briefly, and then provide a number of annotated examples.

3 OML THE LANGUAGE

OML (Optimization Modeling Language) is the modeling language of the Microsoft Solver Foundation.

The basic structure of an OML model is:

```
Model[Parameters[…],Decisions[…],Constraints[…],Goals[…]]
```

Note that OML is case-sensitive throughout, i.e. “Model” is not the same as “model”.

3.1 PARAMETERS

The Parameters section declares sets and parameters. We will discuss here only the declarations. The corresponding data typically is read from a spreadsheet, which we will discuss in section Data Binding.

3.1.1 SETS

Sets are declared in OML but they cannot be populated in OML: all content comes from data binding.

A set is declared as:

```
        Parameters[Sets,Plants,Markets]
```

We now have can use two sets for indexing. The actual elements for these sets will need to come from Excel through data binding (this is explained later).

We can split a declaration in pieces. I.e. the statements

```
        Parameters[Sets,I],Parameters[Sets,J]
```

and

```
        Parameters[Sets,I,J]
```

are identical.

In some cases we want to be able to perform arithmetic on set elements. We can force set elements to be integers using the syntax:

```
        Parameters[Sets[Integers],I,J]
```

3.1.2 SCALARS

A scalar can be declared as follows:

```
        Parameters[Reals,a=10]
```

It is possible to add bounds to the domain. E.g.

```
        Parameters[Reals[0,100],a=110]   // illegal: domain violation
```

will give an error. The same is true for

```
        Parameters[Integers,a=0.5]   // illegal: domain violation
```

The following statements are identical:

```
        Parameters[Reals,a=0.5]   // fraction
```

and

```
        Parameters[Reals,a=1/2]   // alternative notation for fraction
```

3.1.2.1 RESTRICTIONS

Scalars cannot be read from a spreadsheet using the syntax above. If you need to read a scalar from a spreadsheet you need to declare

```
        Parameters[Integer,N[]]   // read scalar from spreadsheet
```

Unfortunately, scalars cannot be assigned a constant expression:

```
        Parameters[Integers,N=10],
        Parameters[Integers,N1=N+1]  // Illegal, no expressions allowed
        Parameters[Integers[0,N],M]  // Illegal, bounds need to be literal numbers
```

You will need to use (N+1) in the model or import N1[] from the spreadsheet.

3.1.3 INDEXED PARAMETERS

Indexed parameters (e.g. vectors and matrices) need to be imported from the spreadsheet. The values cannot be specified directly in OML. For more information see the section on Data Binding.

```
        Parameters[Sets,I,J],
        Parameters[Reals,v[I],A[I,J]],
```

The domain can be tightened if needed:

```
        Parameters[Integers[1,5],w[I]]
```

This will check if the data for w[i] is integer values and between 1 and 5. If the data violates this, an error will be issued and the model will not be solved.

Note: we can use `–Infinity` and `Infinity` in the bound specification.

3.1.3.1 RESTRICTIONS

It is not possible to use a parameter as bound on a domain:

```
        Parameters[Integers,N=10],
        Parameters[Reals[0,N],v[I],A[I,J]],  // Illegal use of N
```

3.2 DECISIONS

The Decisions section declares the variables. It also specifies bounds on the variables and their type (continuous, discrete).

Here are some examples:

```
Decisions[Reals,x],  // a free scalar variable
Decisions[Reals[0,Infinity],y[I,J]], // a non-negative set indexed variable
Decisions[Integers,Foreach[{i,N},d[i]]],
                                // free integer variables d[0],…,d[N-1]
```

If you don't specify bounds OML will assume you deal with free variables. This is like GAMS but unlike many other modeling systems that have non-negative variables as the default.

3.2.1 DOMAINS

There are the following types (domains):

- Reals: for continuous variables
- Integers: for integer and binary variables. If you need to use binary variables or zero-one variables use a lower bound of zero and an upper bound of one: `Decisions[Integers[0,1],x[I,J]]`[4].
- Boolean. These are variables indicating true or false. Note, that these are not the same as binary variables. In many cases one would prefer to use a binary variable as they allow numerical expressions in equations.

3.2.2 FOREACH

The Foreach construct has a number of forms:

- `Foreach[{i,N},…]` loops i=0,1,..N-1. Note that the loop is zero-based.
- `Foreach[{j,k,N},…]` which loops j=k,k+1,..,N-1. Note that the loop does not include N.
- `Foreach[{i,I},…]` loops over set I.

Note that `Foreach[{i,N},…]` is identical to `Foreach[{i,0,N},…]`.

The Foreach construct can be used to declare indexed variables one by one:

```
Parameters[Integers,N=10],
Decisions[Reals, Foreach[{i,N},x[i]]]
```

3.2.3 FILTEREDFOREACH

The FilteredForeach construct is an extension of the Foreach expression: it adds a condition over which the loop is executed. I.e. FilteredForeach[{i,N},i!=3,…] loops i=0,1,2,4,5,…,N-1.

In some cases you may want to generate not a full x[i,j] but only a subset, e.g. only x[i,j] for i>j. This can be specified as:

```
Parameters[Integers,N=3],
Decisions[Reals, Foreach[{i,N},{j,i},x[i,j]]],
                    // strictly lower-triangular matrix
```

If N=3, this will generate the variables x[1,0],x[2,0],x[2,1].

If you would like to generate x[i,j], i=1,...,N, j≤i, (lower triangular including diagonal, one-based indexing) then we have

```
        Decisions[Reals, Foreach[{i,1,N+1},{j,1,i+1},x[i,j]]
```

An alternative formulation would be:

```
        Decisions[Reals,Foreach[{i,1,N+1},FilteredForeach[{j,1,N+1},j<=i,x[i,j]]]],
        // does not work in 1.1 (worked in 1.0)
```

Note that you cannot write:

```
        Decisions[Reals,FilteredForeach[{i,1,N+1},{j,1,N+1},j<=i,x[i,j]]], // illegal
```

In general OML models look better and are simpler if zero-based indexing is used.

The `FilteredForeach[]` construct does not allow a condition to depend on a variable. If a constraint is to be activated depending on a variable, the `Implies[]` construct can be used.

3.3 CONSTRAINTS

Constraints are equations (equalities or inequalities) that put extra conditions on the solution. I often will use the terms constraint, equation and row interchangeably, all indicating the same concept.

The transportation model from section 1 has the following constraint section:

```
Constraints[
      TotalCost == Sum[{i,Plants},{j,Markets},Cost[i,j]*x[i,j]],
      Foreach[{i,Plants}, Sum[{j,Markets},x[i,j]]<=Capacity[i]],
      Foreach[{j,Markets}, Sum[{i,Plants},x[i,j]]>=Demand[j]]
  ],
```

Figure 10. Constraints from the TRNSPORT model

This example directly corresponds to the underlying mathematical model:

$$
\begin{aligned}
TotalCost &= \sum_{i,j} Cost_{i,j} \cdot x_{i,j} \\
\sum_j x_{i,j} &\le Capacity_i \quad \forall i \\
\sum_i x_{i,j} &\ge Demand_j \quad \forall j
\end{aligned}
$$

For a linear programming or mixed-integer programming model, all these constraints have to be linear.

3.3.1 SUMMATIONS

The Sum expression is an important tool in modeling optimization models. Summations over set indexed expressions are very straightforward: `Sum[{i,I},…]`. A double summation in this context is also obvious: `Sum[{i,I},{j,J},…]`.

The summation over integers is like the Foreach. `Sum[{i,N},…]` is summing over i=0,1,…,N-1. The version with a lower limit `Sum[{j,k,N},…]` will sum over j=k,k+1,…,N-1. This means that `Sum[{k,1,3},1] = 2`.

In addition there is a `FilteredSum[]` variant, which adds a condition to the summation. E.g. `FilteredSum[{k,1,4},k!=2,k] = 4`. The `FilteredSum[]` construct does not allow that a condition depends on a variable, even in case of a CSP model. Typically in that case one would use an ordinary Sum with a factor `AsInt[condition]`.

3.3.2 ALL-DIFFERENT CONSTRAINTS

In some constraint programming problems, the following condition may appear:

X[i] = k, where *k* is different for each *i*

This is difficult to model efficiently in a MIP (Williams & Yan, 2001), but can be easily expressed using the `Unequal` construct. The above condition can be stated as:

```
Unequal[Foreach[{i,N},x[i]]]
```

3.3.3 IMPLIES

The construct "if (condition) then constraint" can be formulated using `Implies`:

```
Implies[x==1,y<=10]
```

This is only available for CSP models. Example:

```
Foreach[{i,Tiles},Implies[b[i],x[i]<=N-Side[i] & y[i]<=N-Side[i]]]
```

3.3.4 AND, OR

In CSP models we can use `And[exp1,exp2,…]` and `Or[exp1,exp2,…]`. They can also be written using infix operators: `exp1 & exp2` and `exp1 | exp2`. The example in the previous paragraph shows an application. Another example is:

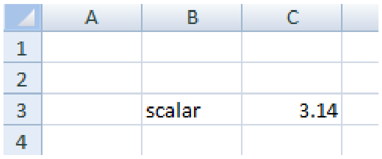
```
Foreach[{i,T},{j,i+1,T},Implies[b[i]&b[j],
                x[i]>=x[j]+Side[j]  |
                x[i]+Side[i]<=x[j]  |
                y[i]>=y[j]+Side[j]  |
                y[i]+Side[i]<=y[j]
           ]]
```

3.4 DATA BINDING

The OML language requires that all data is specified outside the model (except for an occasional scalar constant). The concept to move data from the spreadsheet to the model instance is called Data Binding.

3.4.1 SCALARS

To import a scalar we just need to know the address of the cell where it is located:



	A	B	C
1			
2			
3		scalar	3.14
4			

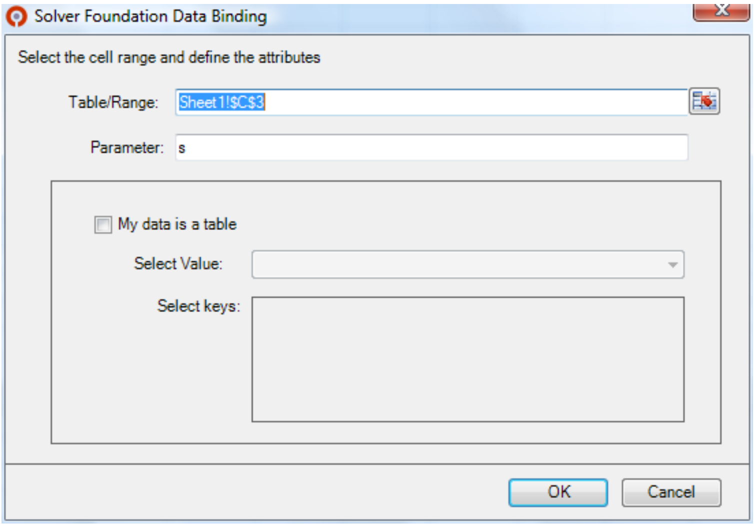
Figure 11. A scalar parameter

In this case this is Sheet1!C3.

In the OML model we specify:

```
Parameters[Reals,s[]],
```

where it is important to use [] in the declaration. [] indicates: the symbol has zero dimensions. Then in the Data Binding dialog we can use:



Solver Foundation Data Binding

Select the cell range and define the attributes

Table/Range: Sheet1!$C$3

Parameter: s

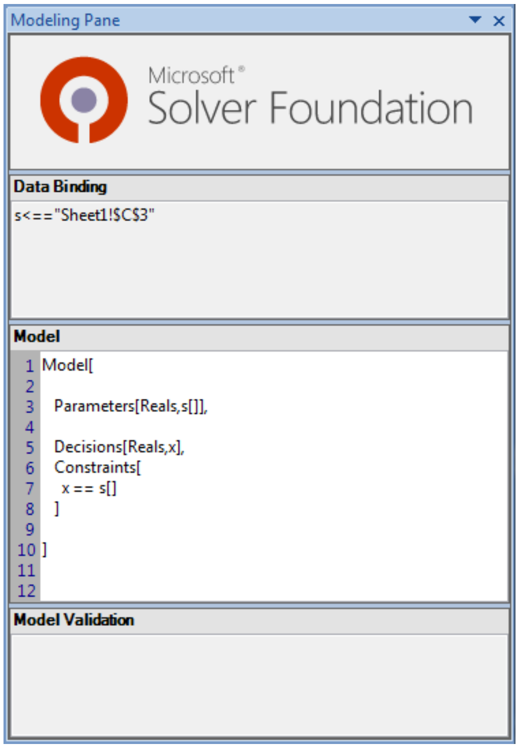
☐ My data is a table

Select Value:

Select keys:

Figure 12. Data binding for a scalar

Note: we did not check “My data is a table”. To test this we can use the minimal model:



```
s<=="Sheet1!$C$3"
```

```
Model[

  Parameters[Reals,s[]],

  Decisions[Reals,x],
  Constraints[
    x == s[]
  ]

]
```

Figure 13. A test model

Note that we have to use a constraint. My first attempt was to write this as a fixed variable:

```
Model[

   Parameters[Reals,s[]],

   Decisions[Reals[s[],s[]],x]  // not allowed

]
```

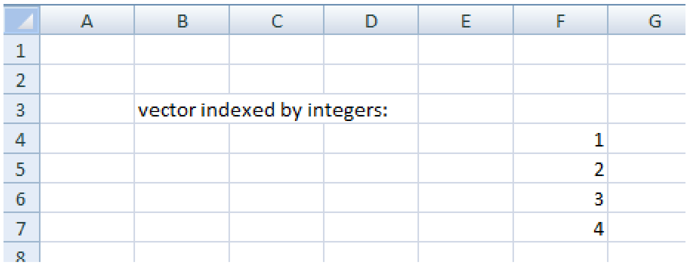
This construct is not allowed: we cannot fix the variable by setting its lower and upper bound to s[].[5]

We can dynamically size a variable based on a bound parameter:

```
Model[

  Parameters[Integers,n[]],  // retrieve through data binding
  Decisions[Reals,Foreach[{i,n[]},x[i]]]  // OK since 1.1

]
```

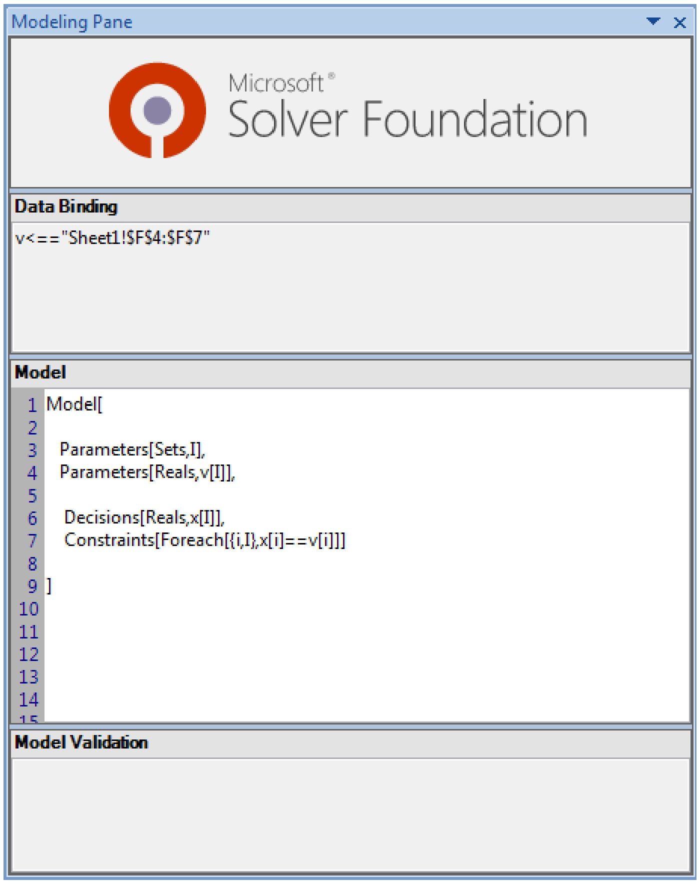
3.4.2 VECTORS

A vector can be imported as follows. The range will become not a single cell but a range indicating the complete vector:



	A	B	C	D	E	F	G
1							
2							
3		vector indexed by integers:					
4						1	
5						2	
6						3	
7						4	
8							

In this case that will be Sheet1!F4:F7. A minimal model to read this is:



```
v<=="Sheet1!$F$4:$F$7"
```

```
Model[

  Parameters[Sets,I],
  Parameters[Reals,v[I]],

   Decisions[Reals,x[I]],
   Constraints[Foreach[{i,I},x[i]==v[i]]]

]
```

The set I will automatically be populated with the index positions 0,1,2,3. This can be seen in the solution report:

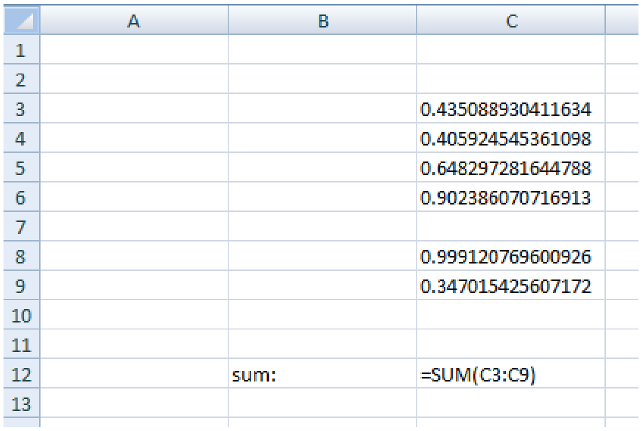
Solver Foundation Results

Name	Value
Solution Type	Optimal
x[0]	1
x[1]	2
x[2]	3
x[3]	4

3.4.3 EMPTY CELLS

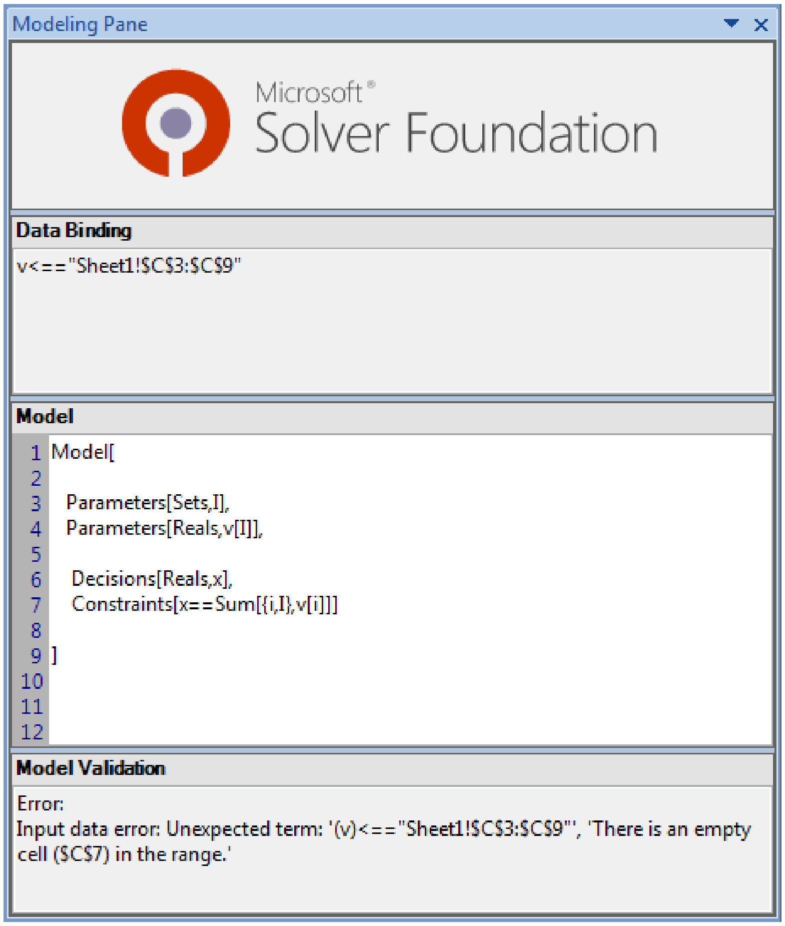
In Excel we often can work with empty cells. E.g. the =SUM formula will not be bothered by empty cells, they will just be skipped. In OML, empty cells cause an error.

E.g. in Excel we can do:



	A	B	C
1			
2			
3			0.435088930411634
4			0.405924545361098
5			0.648297281644788
6			0.902386070716913
7			
8			0.999120769600926
9			0.347015425607172
10			
11			
12		sum:	=SUM(C3:C9)
13			

Excel is fine with the empty cell at C7 but if we try to emulate this in OML we see:



```
v<=="Sheet1!$C$3:$C$9"
```

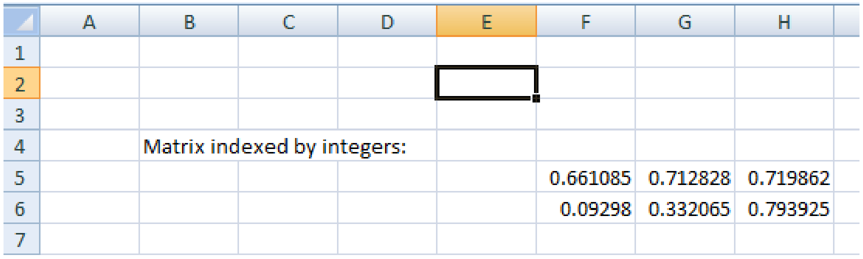
```
Model[

  Parameters[Sets,I],
  Parameters[Reals,v[I]],

   Decisions[Reals,x],
   Constraints[x==Sum[{i,I},v[i]]]

]
```

Error:
Input data error: Unexpected term: '(v)<=="Sheet1!$C$3:$C$9"', 'There is an empty cell ($C$7) in the range.'

As we will see in the examples this is can cause some headaches when dealing with larger models.

3.4.4 MATRICES

This approach extends to matrices. For the data:



	A	B	C	D	E	F	G	H
1								
2								
3								
4		Matrix indexed by integers:						
5						0.661085	0.712828	0.719862
6						0.09298	0.332065	0.793925
7								

we can use the range Sheet1!F5:H6. The model needs to be changed to:

```
Model[

    Parameters[Sets,I,J],
    Parameters[Reals,v[I,J]],

    Decisions[Reals,x[I,J]],
    Constraints[Foreach[{i,I},{j,J},x[i,j]==v[i,j]]]

]
```

In the data binding dialog box again make sure that “My data is a table” is not checked. The solution will now look like:

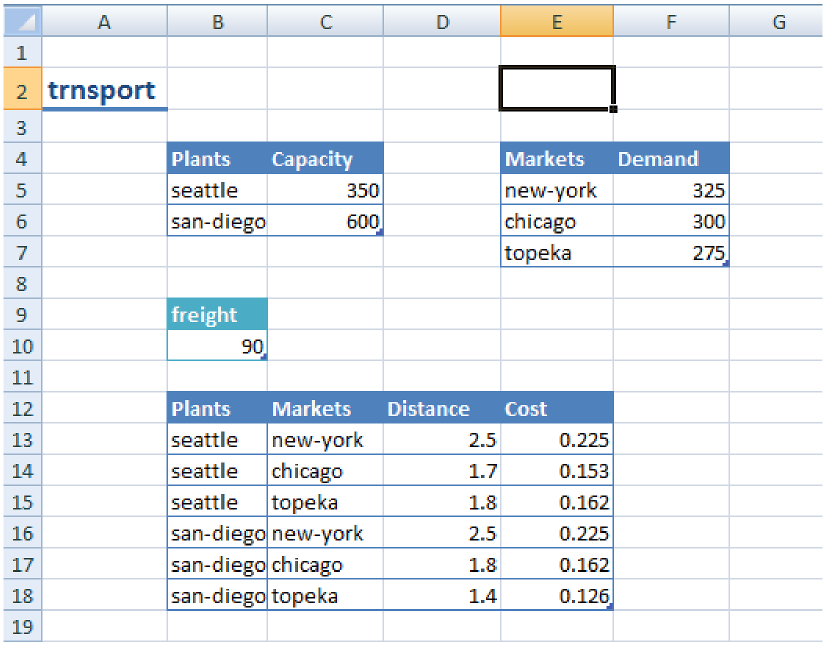
Solver Foundation Results

Name	Value
Solution Type	Optimal
x[0,0]	0.661085096
x[0,1]	0.712827648
x[0,2]	0.719861886
x[1,0]	0.092979643
x[1,1]	0.332064994
x[1,2]	0.793925364

Note again that empty cells are not allowed in OML: you need to specify zero’s explicitly in all empty cells.

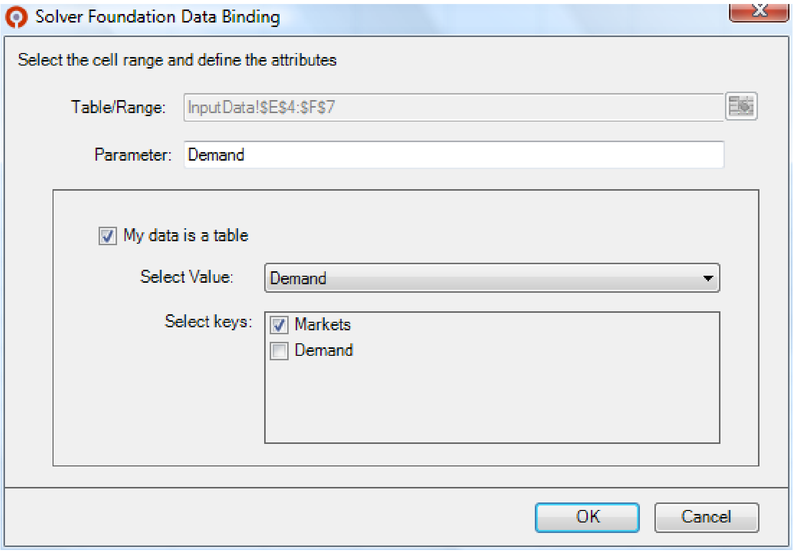
3.4.5 DATA TABLES

Data formatted as a database table can be read using the “My data is a table” option. This will allow you to specify keys (they will becomes index sets in OML) and values. For an example we go back to our transportation model from section 2.2.3. We see there:



	A	B	C	D	E	F	G
1							
2	trnsport						
3							
4		Plants	Capacity		Markets	Demand	
5		seattle	350		new-york	325	
6		san-diego	600		chicago	300	
7					topeka	275	
8							
9		freight					
10		90					
11							
12		Plants	Markets	Distance	Cost		
13		seattle	new-york	2.5	0.225		
14		seattle	chicago	1.7	0.153		
15		seattle	topeka	1.8	0.162		
16		san-diego	new-york	2.5	0.225		
17		san-diego	chicago	1.8	0.162		
18		san-diego	topeka	1.4	0.126		
19							

All data is organized in tables where each column corresponds to a "database field" and each row is a record. In this case the Data Binding dialog will look like:



Solver Foundation Data Binding

Select the cell range and define the attributes

Table/Range: InputData!$E$4:$F$7

Parameter: Demand

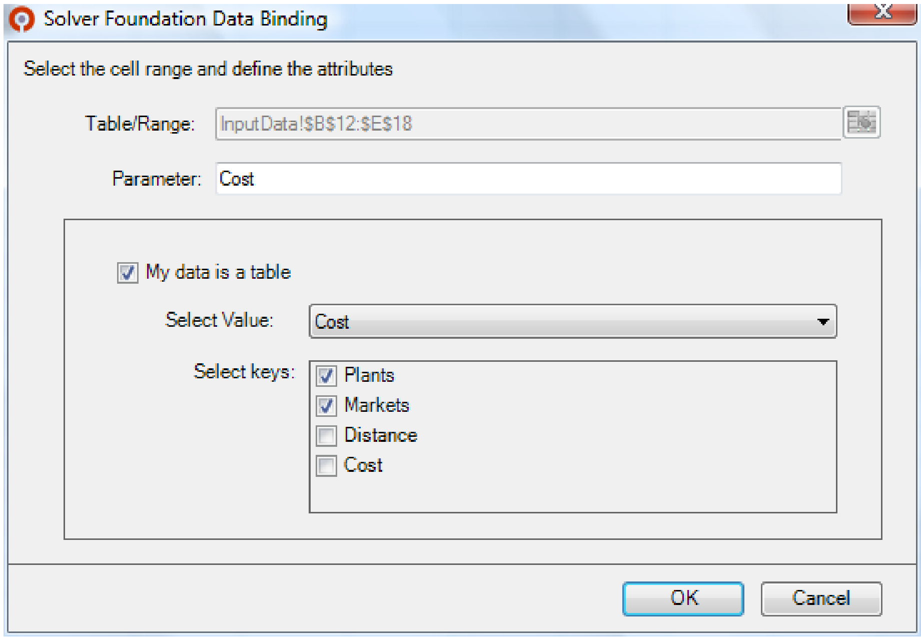
☑ My data is a table

Select Value: Demand

Select keys:
☑ Markets
□ Demand

I.e. in the Demand table, the values are in column Demand (that will also become the name of the parameter) and the keys (set elements) are in column Markets.

From a larger table we can select a Value column. E.g. in the table with Distance and Cost data we want to extract the parameter Cost[Plants,Markets]. This can be achieved by:



Solver Foundation Data Binding

Select the cell range and define the attributes

Table/Range: InputData!$B$12:$E$18

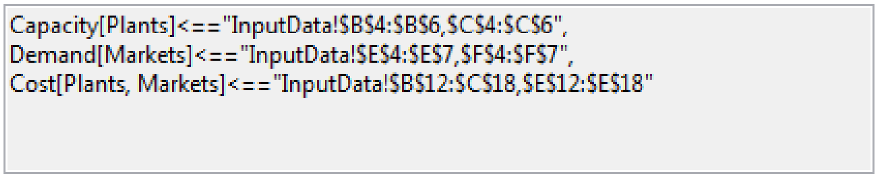
Parameter: Cost

☑ My data is a table

Select Value: Cost

Select keys:
☑ Plants
☑ Markets
☐ Distance
☐ Cost

Note that in the dialog we specified the whole range B12:E18. The system will ignore unused parts (like the column Distance), and it will show the actual used range as:



```
Capacity[Plants]<=="InputData!$B$4:$B$6,$C$4:$C$6",
Demand[Markets]<=="InputData!$E$4:$E$7,$F$4:$F$7",
Cost[Plants, Markets]<=="InputData!$B$12:$C$18,$E$12:$E$18"
```

Indeed column D is not used in the data binding for Cost[Plants,Markets].

From this example you can also see that we used the sets Plants and Markets several times. So who determines what these sets will contain. The answer is that they will contain the union of all elements used.

Note that sets imported this way need to be strings. If the cell has a number OML will complain about a conversion error.

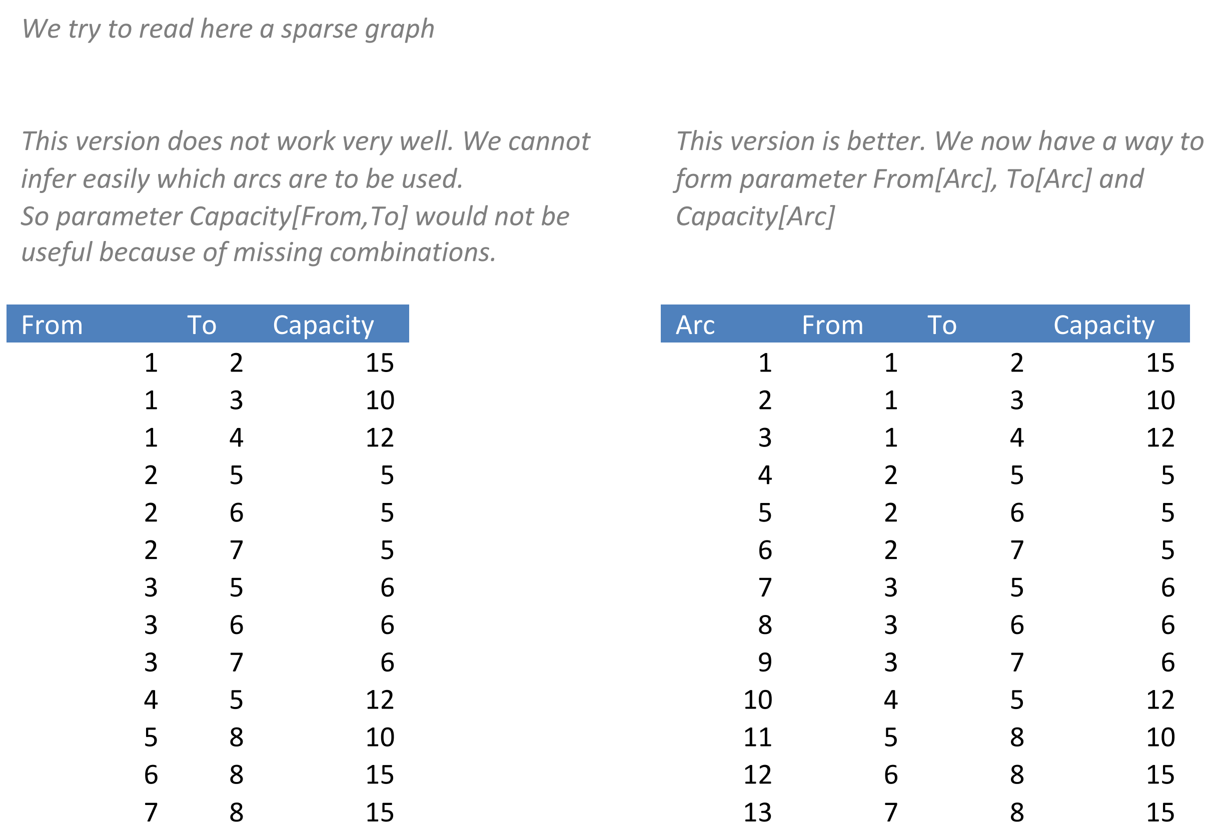
3.4.6 SPARSE TABLES

OML does not support sparse table. That means that records with a zero value need to be explicitly part of the table.

3.4.7 DATA LAYOUT

It is important to think carefully about the layout of your data. Below is an example:

Sparse table



We try to read here a sparse graph

This version does not work very well. We cannot infer easily which arcs are to be used.
So parameter Capacity[From,To] would not be useful because of missing combinations.

This version is better. We now have a way to form parameter From[Arc], To[Arc] and Capacity[Arc]

From	To	Capacity
1	2	15
1	3	10
1	4	12
2	5	5
2	6	5
2	7	5
3	5	6
3	6	6
3	7	6
4	5	12
5	8	10
6	8	15
7	8	15

Arc	From	To	Capacity
1	1	2	15
2	1	3	10
3	1	4	12
4	2	5	5
5	2	6	5
6	2	7	5
7	3	5	6
8	3	6	6
9	3	7	6
10	4	5	12
11	5	8	10
12	6	8	15
13	7	8	15

The left table describes a sparse network. Only some arcs (i,j) exist and have a capacity. OML cannot handle this data. If we would declare `Parameters[Reals, Capacity[From,To]]` then OML assumes all combinations (From,To) exist.

In GAMS we could form the arcs from the sparse capacity table as follows:

```
set arcs(i,j);
arcs(i,j)$capacity(i,j) = yes;
```

i.e. an arc exists where we have a capacity.

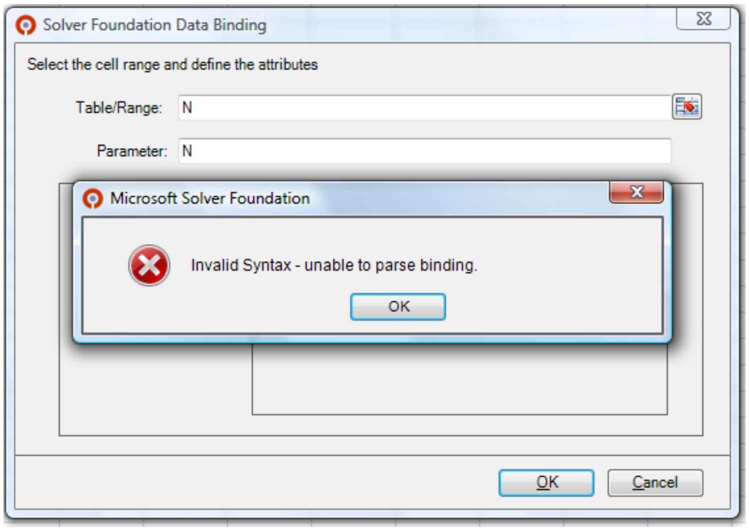
In OML we cannot build any sets or parameters from other sets and parameters. One way to organize the data so an OML model can work with it is to have a complete matrix Capacity[From,To] for all combinations (From.To) with explicitly Capacity[From,To]=0 when the arc does not exist. For larger instances this may become tedious, especially when the graph is very sparse.

Another approach is shown in the right table. Explicitly introduce arc numbers so we can form the parameter Capacity[Arc]. The node-arc incidence can then be formed by parameters From[Arc] and To[Arc].

The exact form of the input data is important not in small part because OML has no data manipulation facilities allowing the modeler to repair or massage ill-fitting data.

3.4.8 RANGE NAMES

Excel data binding does support range names, although the feedback from the GUI does imply otherwise. Excel range names can be global for the whole workbook or work-sheet specific. Even for global names we need to specify the sheet name. E.g. if a range N is used to indicate a scalar parameter, we need to specify: Sheet1!N. If we try to specify only N we see:



Solver Foundation Data Binding

Select the cell range and define the attributes

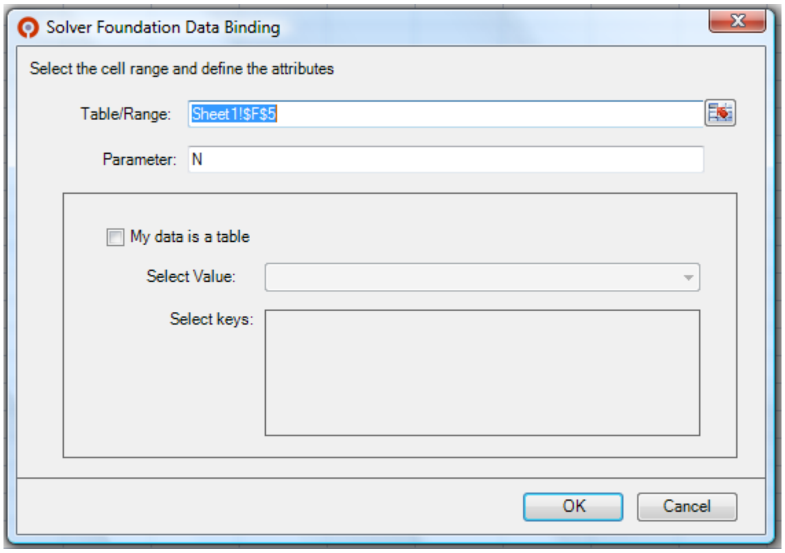
Table/Range: N

Parameter: N

Microsoft Solver Foundation

Invalid Syntax - unable to parse binding.

If we specify Sheet1!N it works. However when we open the binding dialog again, we see:



Solver Foundation Data Binding

Select the cell range and define the attributes

Table/Range: Sheet1!$F$5

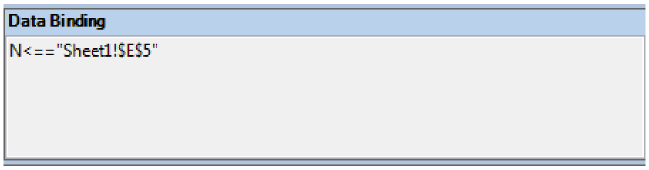
Parameter: N

□ My data is a table

Select Value:

Select keys:

This is misleading however: *the actual underlying range is not Sheet1!$F$5 but rather Sheet1!N.* We can see this if we change the definition of the range N. If we change the range N to say *=Sheet1!$E$5* then (only after pressing the Solve button) we see the data binding info is changed:



Data Binding

N<=="Sheet1!$E$5"

I have not been able to use dynamic ranges with data binding. An example of such a dynamic range is `=OFFSET($A$1,0,0,COUNT($A:$A),1)`. These constructs are used to make ranges automatically expand or contract depending on the size of a table. This is convenient if you want to allow a user to add or delete rows, such that the rest of the spreadsheet is updated accordingly. With fixed ranges, adding a row may not change the actual calculations as the new row is outside the range that is being operated upon. Allowing dynamic ranges makes it possible to create better scalable spreadsheet applications. Of course it is always possible to have some VBA code that manipulates a named range.

3.4.9 DEBUGGING

There is no good facility to debug data binding: there is no display statement and no equation listing or expand command to look at equations. However the following works: introduce extra variables and extra constraints so the variables are equal to the imported data:

```
Foreach[{i,I},xdbug[i]==testpar[i]]
```

Then in the solution report you can inspect the variable.

3.4.10 PERFORMANCE

It looks like the MS Solver Foundation Excel Plug-in is somewhat slow for importing larger data sets (we observed this in this note). Here are a few timings that confirm this:

Cells	MSF Excel Plugin	GAMS gdxxrw
n=100	0.7	1.2
n=1000	2.7	1.3
n=10000	23.0	1.4

OML Model

```
Binding:  P<=="Sheet1!$B$2:$CW$101"
Model[
Parameters[Sets,I,J],
Parameters[Reals,P[I,J]],
Decisions[Reals,d],
Constraints[d==0]
]
```

GAMS Model

```
$call gdxxrw i=data.xlsx par=p rng=A1:CW101
parameter p(*,*);
$gdxin data
$load p

scalar n;
n=card(p);
display n;
```

The MSF code contains a large parameter that is read through data binding, together with a minimal (one variable, one constraint) model. The GAMS code calls GDXXRW to read the spreadsheet and then the intermediate GDX file is read again by GAMS. We display the cardinality of p as a check we read all numbers.

4 SOME API NOTES

I don't want to delve very deep into the API's offered by MSF. In general my view is that modeling in a modeling language is often to be preferred to writing a "matrix generator" in a traditional programming language. However it is useful to discuss some practical cases where we blur the difference a bit: OML can be used from within C# to handle cases where the Excel plug-in has limitations.

4.1 RUNNING OML FROM C#

A simple way to run a scalar OML model embedded in a C# file is as follows.

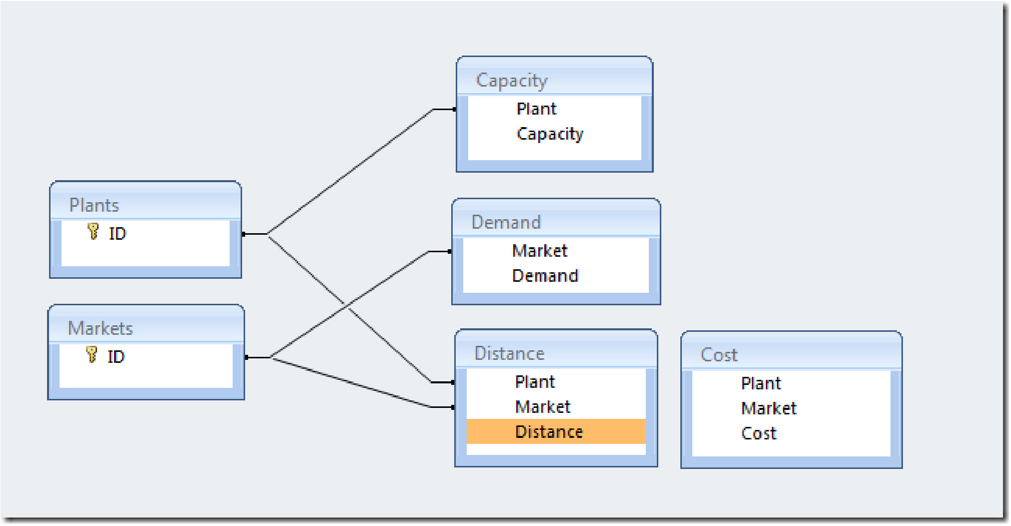
We consider again the simple transportation model from section 2.2. In OML the model looks like:

```
Model[
  Parameters[Sets,Plants,Markets],
  Parameters[Reals,Capacity[Plants],Demand[Markets],Cost[Plants,Markets]],

  Decisions[Reals[0,Infinity],x[Plants,Markets],TotalCost],

  Constraints[
     TotalCost == Sum[{i,Plants},{j,Markets},Cost[i,j]*x[i,j]],
     Foreach[{i,Plants}, Sum[{j,Markets},x[i,j]]<=Capacity[i]],
     Foreach[{j,Markets}, Sum[{i,Plants},x[i,j]]>=Demand[j]]
  ],

  Goals[Minimize[TotalCost]]
]
```

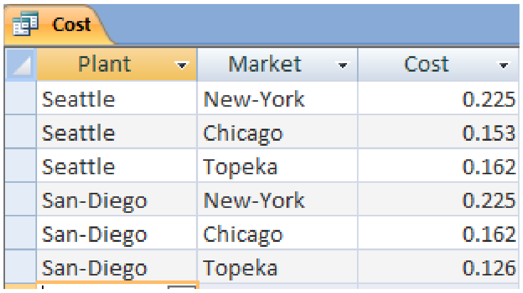
This can be used from C# as follows:

```
        /// <summary>
        /// Holds the OML model
        /// </summary>
        string strModel = @"Model[
              Parameters[Sets,Plants,Markets],
              Parameters[Reals,Capacity[Plants],Demand[Markets],Cost[Plants,Markets]],

              Decisions[Reals[0,Infinity],x[Plants,Markets],TotalCost],

              Constraints[
                 TotalCost == Sum[{i,Plants},{j,Markets},Cost[i,j]*x[i,j]],
                 Foreach[{i,Plants}, Sum[{j,Markets},x[i,j]]<=Capacity[i]],
                 Foreach[{j,Markets}, Sum[{i,Plants},x[i,j]]>=Demand[j]]
              ],

              Goals[Minimize[TotalCost]]
           ]";
        SolverContext context;
        context.LoadModel(FileFormat.OML, new StringReader(strModel));
        Solution solution = context.Solve();
        Console.Write("{0}", solution.GetReport());
```

The real issue is how to handle the data. For this example we stored the data in an Access database as follows:

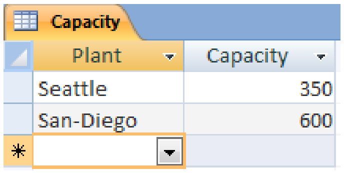


We will use the tables Capacity and Demand and the Query Cost. The data looks like:



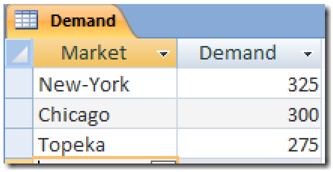
Cost

Plant	Market	Cost
Seattle	New-York	0.225
Seattle	Chicago	0.153
Seattle	Topeka	0.162
San-Diego	New-York	0.225
San-Diego	Chicago	0.162
San-Diego	Topeka	0.126



Capacity

Plant	Capacity
Seattle	350
San-Diego	600



Demand

Market	Demand
New-York	325
Chicago	300
Topeka	275

The reason to choose Access is that Access is simple database. Once we have our code working with Access, we can be reasonable sure that handling other database systems is easy. Essentially we are aiming to handle the least common denominator. The code to handle this can be as follows:

```
        /// <summary>
        /// Solve the problem
        /// </summary>
        public void Solve()
        {
            context.LoadModel(FileFormat.OML, new StringReader(strModel));

            conn = new OleDbConnection(connection);
            foreach (Parameter p in context.CurrentModel.Parameters)
            {
                switch (p.Name)
                {
                    case "Capacity":
                        setBinding(p,"select plant,capacity from capacity",
                            "capacity",new string[]{"plant"});
                        break;
                    case "Demand":
                        setBinding(p,"select market,demand from demand",
                            "demand", new string[]{"market"});
                        break;
                    case "Cost":
                        setBinding(p,"select plant,market,cost from cost",
                            "cost", new string[]{"plant", "market"});
                        break;

                }

            }

            Solution solution = context.Solve();
            Console.Write("{0}", solution.GetReport());

        }
```

In each binding operation we specify:

1. The SFS parameter, which we retrieve from the CurrentModel
2. The query to be used against the database
3. The name of the data column
4. The names of the index columns (passed on as an array of strings)

The complete model looks like:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data;
using System.Data.OleDb;
using System.Data.Linq;
using System.Text;
using Microsoft.SolverFoundation.Services;
using System.IO;

namespace OML1
{
    class Trnsport
    {
        /// <summary>
        /// Called by the OS
        /// </summary>
        /// <param name="args"></param>
        static void Main(string[] args)
        {
            Trnsport t = new Trnsport();
            t.Solve();
        }

        /// <summary>
        /// Holds the OML model
        /// </summary>
        string strModel = @"Model[
              Parameters[Sets,Plants,Markets],
              Parameters[Reals,Capacity[Plants],Demand[Markets],Cost[Plants,Markets]],

              Decisions[Reals[0,Infinity],x[Plants,Markets],TotalCost],

              Constraints[
                 TotalCost == Sum[{i,Plants},{j,Markets},Cost[i,j]*x[i,j]],
                 Foreach[{i,Plants}, Sum[{j,Markets},x[i,j]]<=Capacity[i]],
                 Foreach[{j,Markets}, Sum[{i,Plants},x[i,j]]>=Demand[j]]
              ],

              Goals[Minimize[TotalCost]]
           ]";

        /// <summary>
        /// Connection string for MS Access
        /// Use x86 architecture!
        /// </summary>
        string connection = @"Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=C:\projects\ms\OML1\OML1\trnsport.accdb;Persist Security Info=False;";

        /// <summary>
        ///  SFS
        /// </summary>
        SolverContext context;

        /// <summary>
        /// One db connection
        /// </summary>
        OleDbConnection conn;

        /// <summary>
        ///  Constructor
        /// </summary>
        public Trnsport()
        {
            context = SolverContext.GetContext();
        }

        /// <summary>
        /// get query result as DataSet
        /// </summary>
        /// <param name="connection">connection string</param>
        /// <param name="query">query as string</param>
        /// <returns></returns>
        private DataSet SelectOleDbSrvRows(string connection, string query)
        {
            DataSet ds = new DataSet();
            OleDbDataAdapter adapter = new OleDbDataAdapter();
            adapter.SelectCommand = new OleDbCommand(query, conn);
            adapter.Fill(ds);
            return ds;
        }
        /// <summary>
        /// Perform some magic to make sure the query output arrives in OML model.
        /// </summary>
        /// <param name="p">OML/SFS parameter</param>
        /// <param name="query">database query</param>
        /// <param name="valueColumn">column with values</param>
        /// <param name="IndexColumns">columns with indices</param>
        private void setBinding(Parameter p, string query, string valueColumn, string[]
IndexColumns)
        {
             DataSet ds = SelectOleDbSrvRows(connection, query);
             DataTable dt = ds.Tables[0];
             p.SetBinding(dt.AsEnumerable(), valueColumn, IndexColumns);
        }

        /// <summary>
        /// Solve the problem
        /// </summary>
        public void Solve()
        {
            context.LoadModel(FileFormat.OML, new StringReader(strModel));

            conn = new OleDbConnection(connection);

            foreach (Parameter p in context.CurrentModel.Parameters)
            {
                switch (p.Name)
                {
                    case "Capacity":
                        setBinding(p,"select plant,capacity from capacity",
                            "capacity",new string[]{"plant"});
                        break;
                    case "Demand":
                        setBinding(p,"select market,demand from demand",
                            "demand", new string[]{"market"});
                        break;
                    case "Cost":
                        setBinding(p,"select plant,market,cost from cost",
                            "cost", new string[]{"plant", "market"});
                        break;

                }

            }

            Solution solution = context.Solve();
            Console.Write("{0}", solution.GetReport());

        }

    }
}
```

4.2 CALLING C# FROM EXCEL

Sometimes models are not suited to be expressed in OML only. An example would be because we need to solve different models or need to implement an algorithm. In this case we really need to use the Solver Foundation API's. Can we still use Excel as front-end to host the user-interface? The answer is yes. There are basically two ways to call .Net in general or C# specifically from Excel. The first one is to create a .Net DLL with appropriate COM interfaces. Then this DLL can be called from VBA. This approach is illustrated in section **Error! Reference source not found.**. Another approach is to use VSTO: a framework to call .Net from Office applications. This is demonstrated in section 5.13.

5 EXAMPLES

In this section we will implement a few examples. The first example was a simple linear programming problem based on a transportation problem and was discussed in the introduction section. Here we discuss a few other models.

The example models are not real-life models. We use here smaller artificial models, as they can be explained easier while they can illustrate a certain modeling issue in a more succinct way. Real-life models tend to be large, complicated, and messy and they have lots of extra details that just obscure the issues we want to demonstrate.

5.1 A DIET PROBLEM

The diet problem goes back to work by the economist George Stigler (Stigler, 1945), although earlier formulations have been mentioned (Murphy, 1996). It is one of the first optimization problems to be studied back in the 1930’s and 40’s. It was first motivated by the Army’s desire to meet the nutritional requirements of the field GI’s while minimizing the cost.

Below we show a part of the table that gives nutrient contents of different commodities per dollar spent.

id	description	units	price	weight	calories	protein	calcium	iron	vitamin A
			aug 15 1939	*edible per $1*					
			(cents)	*(grams)*	*(1000)*	*(grams)*	*(grams)*	*(mg.)*	*(1000 IU)*
flour	Wheat Flour (Enriched)	10 lb.	36	12600	44.7	1411	2	365	
macaroni	Macaroni	1 lb.	14.1	3217	11.6	418	0.7	54	
cereal	Wheat Cereal (Enriched)	28 oz.	24.2	3280	11.8	377	14.4	175	
cornflakes	Corn Flakes	8 oz.	7.1	3194	11.4	252	0.1	56	
cornmeal	Corn Meal	1 lb.	4.6	9861	36	897	1.7	99	30.9
grits	Hominy Grits	24 oz.	8.5	8005	28.6	680	0.8	80	
rice	Rice	1 lb.	7.5	6048	21.2	460	0.6	41	

In addition we have a table of minimum requirements per day:

recommended daily allowances for a moderately active man	
calories	3
protein	70
calcium	0.8
iron	12
vitaminA	5
thiamine	1.8
riboflavin	2.7

niacin	18
ascorbicAcid	75

Together we can compute the least expensive diet, to keep the poor soul subject to this diet-plan alive. The linear programming model is fairly straightforward:

$$\min \sum_{c \in C} Buy_c$$
$$\sum_{c \in C} Content_{c,n} Buy_c \geq Allowance_n \;\; \forall n \in N$$

This is very simple:

```
Model[
   Parameters[Sets,C,N],
   Parameters[Reals,content[C,N],allowance[N]],

   Decisions[Reals[0,Infinity],TotalCost,Buy[C]],
   Constraints[
     TotalCost == Sum[{c,C},Buy[c]],
     Foreach[{n,N}, Sum[{c,C}, content[c,n]*Buy[c]] >= allowance[n]]
   ],
   Goals[Minimize[TotalCost]]
]
```

The objective is very simple as the variable Buy is expressed in dollars. If the unit was related to volume we would have seen a cost coefficient different from all one's. We have placed the objective in the Constraints section. It is also possible to write:

```
Model[
   Parameters[Sets,C,N],
   Parameters[Reals,content[C,N],allowance[N]],

   Decisions[Reals[0,Infinity],Buy[C]],
   Constraints[
     Foreach[{n,N}, Sum[{c,C}, content[c,n]*Buy[c]] >= allowance[n]]
   ],
   Goals[Minimize[TotalCost -> Sum[{c,C},Buy[c]]]]
]
```

Here we have a real objective function `Sum[{c,C},Buy[c]]` with a label `TotalCost.`

This compares directly to the original GAMS model:

```
positive variable x(c)  'dollars of food to be purchased daily   (dollars)'
;

free variable cost      'total food bill                         (dollars)'

equations
    nb(n)     'nutrient balance   (units)'
    cb        'cost balance       (dollars)'
```

```
;

nb(n).. sum(c, data(c,n)*x(c)) =g= allowance(n);
cb..    cost=e= sum(c, x(c));

model diet 'stiglers diet problem' / nb,cb /;
solve diet minimizing cost using lp;
```

The problem is however with the data. A missing entry for a `Content[c,n]` means it has zero content for that particular nutrient. In GAMS this is handled automatically as its sparse matrix storage implies that non-existent and zero is the same thing.

To be able to import the data in OML and sets we need to reorganize the data substantially. I use the table format:

C	N	Content
Flour	Calories	44.7
Flour	Protein	1411
Flour	Calcium	2
Flour	Iron	365
Flour	vitaminA	0
Flour	Thiamine	55.4
Flour	Riboflavin	33.3
Flour	Niacin	441
Flour	ascorbicAcid	0
Macaroni	Calories	11.6
Macaroni	Protein	418
Macaroni	Calcium	0.7
Macaroni	Iron	54
Macaroni	vitaminA	0
Macaroni	Thiamine	3.2
Macaroni	Riboflavin	1.9
Macaroni	Niacin	68
Macaroni	ascorbicAcid	0

Here we introduced many zeros where needed as OML does not handle sparse tables. Some of the inserted records are highlighted in the above table. We also removed some records (price, weight) that were not required. For larger data sets adding the zero records explicitly may be a lot of work and it may use up a lot of space in the spreadsheet.

In general we need to be very precise with the data to match exactly the parameters and sets in the model. We cannot repair any problems with the data in the OML model. This is very different from GAMS where one often takes the data as is, and then manipulate the data until suitable for use in the model equations.

5.2 MAX FLOW, A NETWORK MODEL

The max flow problem can be stated as:

$$\max f$$

$$\forall i: \sum_{arc\,(j,i)} x_{j,i} - \sum_{arc\,(i,j)} x_{i,j} = \begin{cases} -f & \text{if i is a source} \\ f & \text{if i is a sink} \\ 0 & \text{otherwise} \end{cases}$$

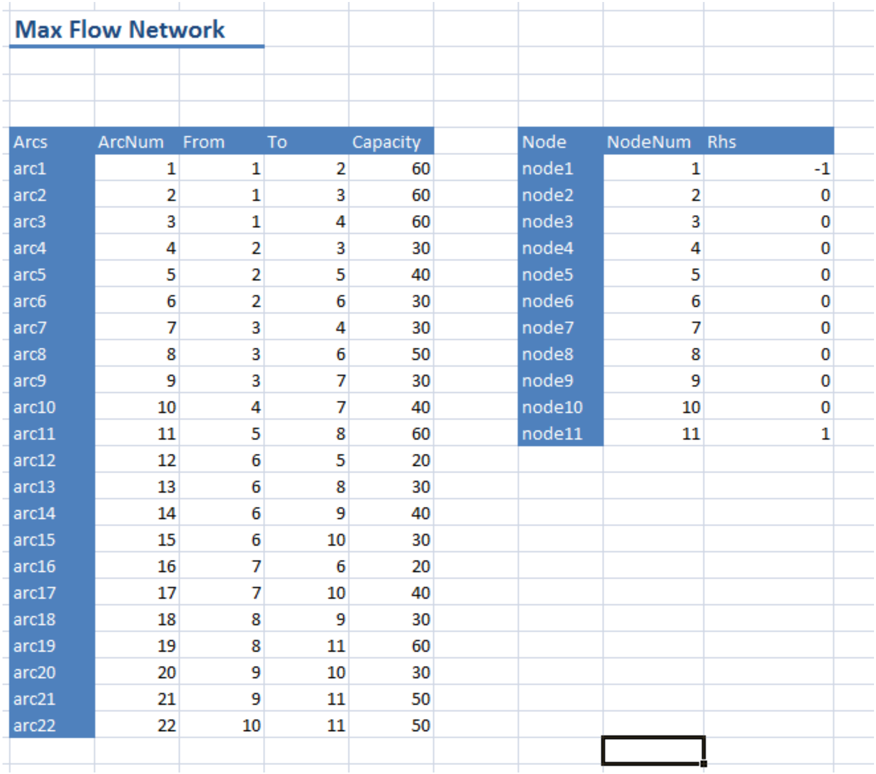
$$0 \le x_{i,j} \le cap(i,j)$$

The model is not too difficult to state in OML. An issue is that many networks are sparse, a concept not really supported by OML. The GAMS formulation can look like:

```
$ontext

   max flow network example

   Data from example in
     Mitsuo Gen, Runwei Cheng, Lin Lin
     Network Models and Optimization: Multiobjective Genetic Algorithm Approach
     Springer, 2008

   Erwin Kalvelagen, Amsterdam Optimization, May 2008

$offtext


sets
   i 'nodes' /node1*node11/
   source(i)    /node1/
   sink(i)     /node11/
;

alias(i,j);

parameter capacity(i,j) /
   node1.node2    60
   node1.node3    60
   node1.node4    60
   node2.node3    30
   node2.node5    40
   node2.node6    30
   node3.node4    30
   node3.node6    50
   node3.node7    30
   node4.node7    40
   node5.node8    60
   node6.node5    20
   node6.node8    30
   node6.node9    40
   node6.node10   30
   node7.node6    20
   node7.node10   40
   node8.node9    30
   node8.node11   60
   node9.node10   30
   node9.node11   50
   node10.node11  50
/;



set arcs(i,j);
arcs(i,j)$capacity(i,j) = yes;
display arcs;
```

```
parameter rhs(i);
rhs(source) = -1;
rhs(sink) = 1;

variables
   x(i,j) 'flow along arcs'
   f      'total flow'
;

positive variables x;
x.up(i,j) = capacity(i,j);

equations
   flowbal(i)  'flow balance'
;

flowbal(i)..   sum(arcs(j,i), x(j,i)) - sum(arcs(i,j), x(i,j)) =e= f*rhs(i);

model m/flowbal/;

solve m maximizing f using lp;
```

We discussed this already in section 3.4.7. The sparse network cannot be handled directly by OML. Of course we could generate a dense graph by introducing capacities for all arcs between any two nodes. For larger networks this is not a reasonable approach. As a workaround, we number the arcs and have as data From[.] and To[.]. This looks like:



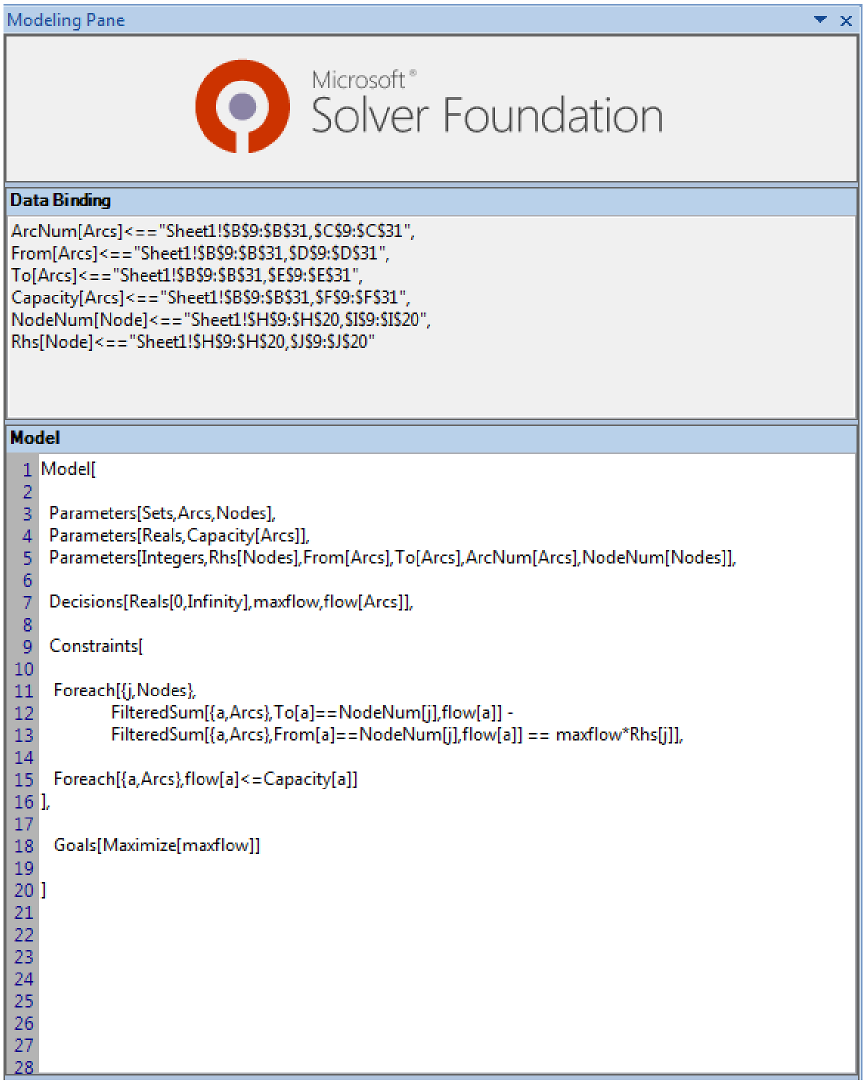
Max Flow Network

Arcs	ArcNum	From	To	Capacity
arc1	1	1	2	60
arc2	2	1	3	60
arc3	3	1	4	60
arc4	4	2	3	30
arc5	5	2	5	40
arc6	6	2	6	30
arc7	7	3	4	30
arc8	8	3	6	50
arc9	9	3	7	30
arc10	10	4	7	40
arc11	11	5	8	60
arc12	12	6	5	20
arc13	13	6	8	30
arc14	14	6	9	40
arc15	15	6	10	30
arc16	16	7	6	20
arc17	17	7	10	40
arc18	18	8	9	30
arc19	19	8	11	60
arc20	20	9	10	30
arc21	21	9	11	50
arc22	22	10	11	50

Node	NodeNum	Rhs
node1	1	-1
node2	2	0
node3	3	0
node4	4	0
node5	5	0
node6	6	0
node7	7	0
node8	8	0
node9	9	0
node10	10	0
node11	11	1

The complete model can look like:

```
Model[

  Parameters[Sets,Arcs,Nodes],
  Parameters[Reals,Capacity[Arcs]],
  Parameters[Integers,Rhs[Nodes],From[Arcs],To[Arcs],ArcNum[Arcs],NodeNum[Nodes]],

  Decisions[Reals[0,Infinity],maxflow,flow[Arcs]],

  Constraints[

   Foreach[{j,Nodes},
       FilteredSum[{a,Arcs},To[a]==NodeNum[j],flow[a]] -
              FilteredSum[{a,Arcs},From[a]==NodeNum[j],flow[a]] == maxflow*Rhs[j]],

   Foreach[{a,Arcs},flow[a]<=Capacity[a]]
],

   Goals[Maximize[maxflow]]

]
```



```
ArcNum[Arcs]<=="Sheet1!$B$9:$B$31,$C$9:$C$31",
From[Arcs]<=="Sheet1!$B$9:$B$31,$D$9:$D$31",
To[Arcs]<=="Sheet1!$B$9:$B$31,$E$9:$E$31",
Capacity[Arcs]<=="Sheet1!$B$9:$B$31,$F$9:$F$31",
NodeNum[Node]<=="Sheet1!$H$9:$H$20,$I$9:$I$20",
Rhs[Node]<=="Sheet1!$H$9:$H$20,$J$9:$J$20"
```

```
Model[

 Parameters[Sets,Arcs,Nodes],
 Parameters[Reals,Capacity[Arcs]],
 Parameters[Integers,Rhs[Nodes],From[Arcs],To[Arcs],ArcNum[Arcs],NodeNum[Nodes]],

 Decisions[Reals[0,Infinity],maxflow,flow[Arcs]],

 Constraints[

  Foreach[{j,Nodes},
           FilteredSum[{a,Arcs},To[a]==NodeNum[j],flow[a]] -
           FilteredSum[{a,Arcs},From[a]==NodeNum[j],flow[a]] == maxflow*Rhs[j]],

  Foreach[{a,Arcs},flow[a]<=Capacity[a]]
],

  Goals[Maximize[maxflow]]

]
```

The node balance equation becomes a little bit unwieldy and less readable with this approach. This is a small, artificial example but note that quite a few models have some network component. It illustrates that leaving out sparse data handling and multidimensional sets to simplify a modeling language, although not a show stopper, comes with a cost in the form of additional complexity for the modeler. Note that in this model we still have a possibly large node table to maintain (the GAMS model handles this sparse).

5.3 THE SOCIAL GOLFER PROBLEM

I am recently involved in a practical application of a scheduling problem related to the Social Golfer Problem. The particular application is somewhat more complicated, but we could use the GAMS/Cplex model described here as a starting point.

The problem is to find a good schedule for a number (N) of golf players. They play T rounds in groups of size GS. I.e. we have T*GS=N. The schedule has to be designed that each golfer meets another golfer at most one time.

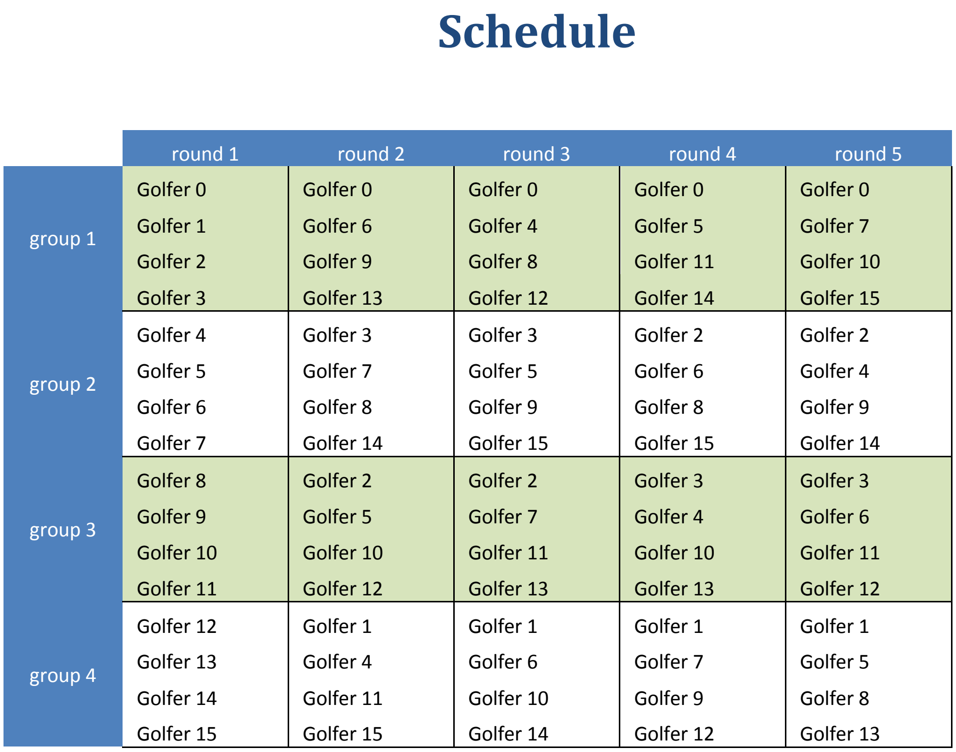
For smaller instances of the pure Social Golfer Problem, it is convenient to use a CSP approach:

variable $x_{i,g,t} \in \{0,1\}$	Binary variable indicating if player *i* is playing in group *g* in round *t*.
$\sum_{g} x_{i,g,t} = 1 \;\; \forall i,t$	A player has to play each round in exactly one group.
$\sum_{i} x_{i,g,t} = GS \;\; \forall g,t$	Each group consist of GS players
$\sum_{g,t} x_{i,g,t} x_{j,g,t} \leq 1 \;\; \forall i,j$	Restrict the number of times players *i* and *j* meet. This is a non-linear constraint, but that is no problem in a CSP setting. The equation is specified here for each combination i and j. Note however that if we compared *i* and *j* we no longer have to inspect *j* and *i*, so we can restrict the number of equations by exploiting symmetry here.

Without loss of generality we can fix the first round and fix the first player. Here is a formulation in OML:

```
Model[

// N : number of golfers
// NG : number of groups
// T : number of rounds
// GS : group size (N/NG)

 Parameters[Integers,N=16,NG=4,T=5,GS=4],
 // Would prefer: GS=N/NG but OML does not allow (constant) expressions
 // in parameter statements

 Decisions[
  Integers[0,1],
   Foreach[{i,N},{g,NG},{t,T},x[i,g,t]]
 ],


 Constraints[
     // each golfer has to play each round
     Foreach[{i,N},{t,T},Sum[{g,NG},x[i,g,t]] == 1],

     // form groups
     Foreach[{g,NG},{t,T},Sum[{i,N},x[i,g,t]] == GS],

     // golfer i and j meet at most one time
     // Foreach[{i,N}, FilteredForeach[{j,N},i!=j, Sum[{g,NG},{t,T},x[i,g,t]*x[j,g,t]] <= 1]],
     // We exploit symmetry here:
     Foreach[{i,N}, Foreach[{j,i+1,N}, Sum[{g,NG},{t,T},x[i,g,t]*x[j,g,t]] <= 1]],


     // fix first round
     Foreach[{g,NG},{k,GS},x[GS*g+k,g,0]==1],

     // fix first golfer
     Foreach[{t,1,T},x[0,0,t]==1]
 ]

]
```

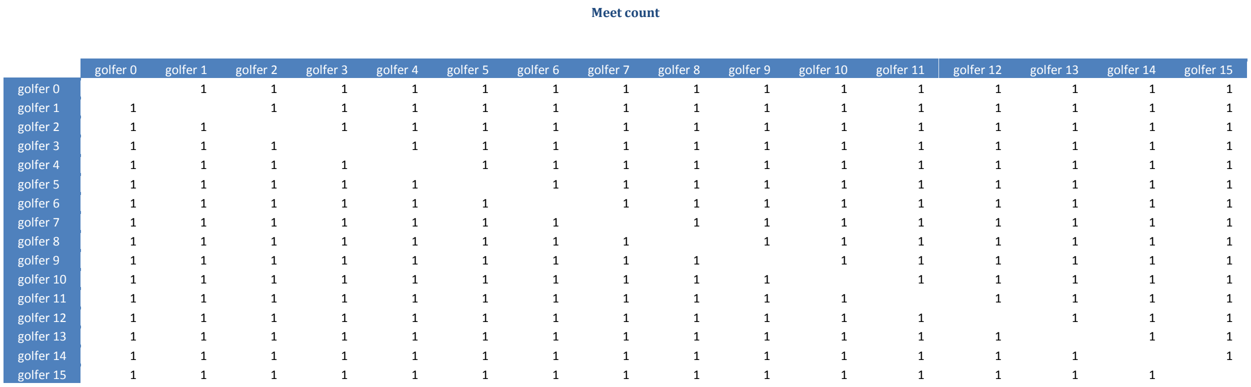
The resulting scheduling looks like:



Schedule

	round 1	round 2	round 3	round 4	round 5
group 1	Golfer 0 Golfer 1 Golfer 2 Golfer 3	Golfer 0 Golfer 6 Golfer 9 Golfer 13	Golfer 0 Golfer 4 Golfer 8 Golfer 12	Golfer 0 Golfer 5 Golfer 11 Golfer 14	Golfer 0 Golfer 7 Golfer 10 Golfer 15
group 2	Golfer 4 Golfer 5 Golfer 6 Golfer 7	Golfer 3 Golfer 7 Golfer 8 Golfer 14	Golfer 3 Golfer 5 Golfer 9 Golfer 15	Golfer 2 Golfer 6 Golfer 8 Golfer 15	Golfer 2 Golfer 4 Golfer 9 Golfer 14
group 3	Golfer 8 Golfer 9 Golfer 10 Golfer 11	Golfer 2 Golfer 5 Golfer 10 Golfer 12	Golfer 2 Golfer 7 Golfer 11 Golfer 13	Golfer 3 Golfer 4 Golfer 10 Golfer 13	Golfer 3 Golfer 6 Golfer 11 Golfer 12
group 4	Golfer 12 Golfer 13 Golfer 14 Golfer 15	Golfer 1 Golfer 4 Golfer 11 Golfer 15	Golfer 1 Golfer 6 Golfer 10 Golfer 14	Golfer 1 Golfer 7 Golfer 9 Golfer 12	Golfer 1 Golfer 5 Golfer 8 Golfer 13

Indeed the meet counts are what we want:



Meet count

	golfer 0	golfer 1	golfer 2	golfer 3	golfer 4	golfer 5	golfer 6	golfer 7	golfer 8	golfer 9	golfer 10	golfer 11	golfer 12	golfer 13	golfer 14	golfer 15
golfer 0		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
golfer 1	1		1	1	1	1	1	1	1	1	1	1	1	1	1	1
golfer 2	1	1		1	1	1	1	1	1	1	1	1	1	1	1	1
golfer 3	1	1	1		1	1	1	1	1	1	1	1	1	1	1	1
golfer 4	1	1	1	1		1	1	1	1	1	1	1	1	1	1	1
golfer 5	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1
golfer 6	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1
golfer 7	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1
golfer 8	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1
golfer 9	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1
golfer 10	1	1	1	1	1	1	1	1	1	1		1	1	1	1	1
golfer 11	1	1	1	1	1	1	1	1	1	1	1		1	1	1	1
golfer 12	1	1	1	1	1	1	1	1	1	1	1	1		1	1	1
golfer 13	1	1	1	1	1	1	1	1	1	1	1	1	1		1	1
golfer 14	1	1	1	1	1	1	1	1	1	1	1	1	1	1		1
golfer 15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

A different formulation would be to use an integer variable x[i,t]=g which corresponds to x[i,g,t]=1 in the above model.

variable $x_{i,t} \in \{1,\ldots,NG\}$ Integer variable indicating in which group golfer *i* plays in round *t*.

$$\sum_{i|x_{i,t}=g} 1 = GS \quad \forall t,g$$

The number of golfers in a group must be NG

$$\sum_{t|x_{i,t}=x_{j,t}} 1 \leq 1 \;\; \forall i,j$$

The number of times golfer *i* and *j* meet. Again we can exploit symmetry here.

The OML representation can look like:

```
Model[

// N : number of golfers
// NG : number of groups
// T : number of rounds
// GS : group size (N/NG)

  Parameters[Integers,N=16,NG=4,T=5,GS=4],
  // Parameters[Integers,N=32,NG=8,T=10,GS=4],

  Decisions[
    //Integers[0,NG-1],
      Integers[0,3],
       Foreach[{i,N},{t,T},x[i,t]]
  ],

  Constraints[
     // form groups
     Foreach[{g,NG},{t,T},Sum[{i,N},AsInt[x[i,t]==g]] == GS],

     // golfer i and j meet at most one time
     Foreach[{i,N}, {j,i+1,N}, Sum[{t,T},AsInt[x[i,t]==x[j,t]]] <= 1],

     // fix first round
     Foreach[{g,NG},{k,GS},x[GS*g+k,0]==g],

     // fix first golfer
     Foreach[{t,1,T},x[0,t]==0]
  ]
]
```

This formulation would not be possible with a MIP solver. The performance of the first formulation seems a little bit better.

In a MIP formulation we can go back to our binary variables x[i,g,t]. The binary multiplication in the meet count equation can be linearized as:

$$\begin{gathered} m_{i,j,g,t} \leq x_{i,g,t} \\ m_{i,j,g,t} \leq x_{j,g,t} \\ m_{i,j,g,t} \geq x_{i,g,t} + x_{j,g,t} - 1 \\ \sum_{g,t} m_{i,j,g,t} \leq 1 \end{gathered}$$

where $0 \le m_{i,j,g,t} \le 1$is a new variable; this variable can be continuous. The first two inequalities can actually be dropped, as we are only interested in keeping the number of m's that are one down. We see that a MIP formulation needs many more equations and variables than a corresponding CSP model.

5.4 JOB SHOP SCHEDULING

Scheduling models are sometimes very difficult to solve as a mathematical programming problems. A good example of this is the standard Job Shop Scheduling problem.

Probably the best way to explain the problem is looking at a data set:

	task1		task2		task3		task4		task5		task6	
	machine	time	machine	time	Machine	time	machine	time	machine	time	machine	time
job1	m2	1	m0	3	m1	6	m3	7	m5	3	m4	6
job2	m1	8	m2	5	m4	10	m5	10	m0	10	m3	4
job3	m2	5	m3	4	m5	8	m0	9	m1	1	m4	7
job4	m1	5	m0	5	m2	5	m3	3	m4	8	m5	9
job5	m2	9	m1	3	m4	5	m5	4	m0	3	m3	1
job6	m1	3	m3	3	m5	9	m0	10	m4	4	m2	1

Each job has to go through a number of stages (called tasks here) on different machines. The times a task occupies a machine is listed in the table. Each machine can only work on one task at the time. The goal is to design a schedule that minimizes the total make span, i.e. the time that the last task is finished.

The basic model looks like:

variable $x_{j,m} \ge 0$

Start time of task of running job *j* on machine *m*

variable $y_{j,k,m} \in \{0,1\}$

A binary variable indicating whether job *j* comes after job *k* on machine *m*

$x_{j,m2} \ge x_{j,m1} + t_{j,m1}$

Precedence equations. For certain combinations *(j,m1,m2)* we need to prescribe a sequencing: first execute task on machine *m* before we can execute on machine *m2*. The sequencing data is taken from the table above. E.g. $x_{job\ 1,mac\ hine\ 0} \ge x_{job\ 1,mac\ hine\ 2} + 1$. We only need to do this for tasks that immediately follow each other. This is not very easy to do in OML as we don't have sparse multi-dimensional sets.

$x_{j,m} \ge x_{k,m} + t_{k,m} - My_{j,k,m}$
$x_{k,m} \ge x_{j,m} + t_{j,m} - M(1 - y_{j,k,m})$

No overlap equations. These equations make sure a machine is occupied by only up to one job at the time. This is modeled by an 'or' condition: job *j* is executed on machine *m* before or after job *k*. This is a big-M formulation: we need to find an appropriate value for it. In the model we just use the sum of all processing times for this: no job will be scheduled later than that. This type of constraint is typical in scheduling applications.

minimize z
$z \geq x_{j,m} + t_{j,m}$

The objective is to minimize the total make span. This is modeled by minimizing the finishing time of the last task.

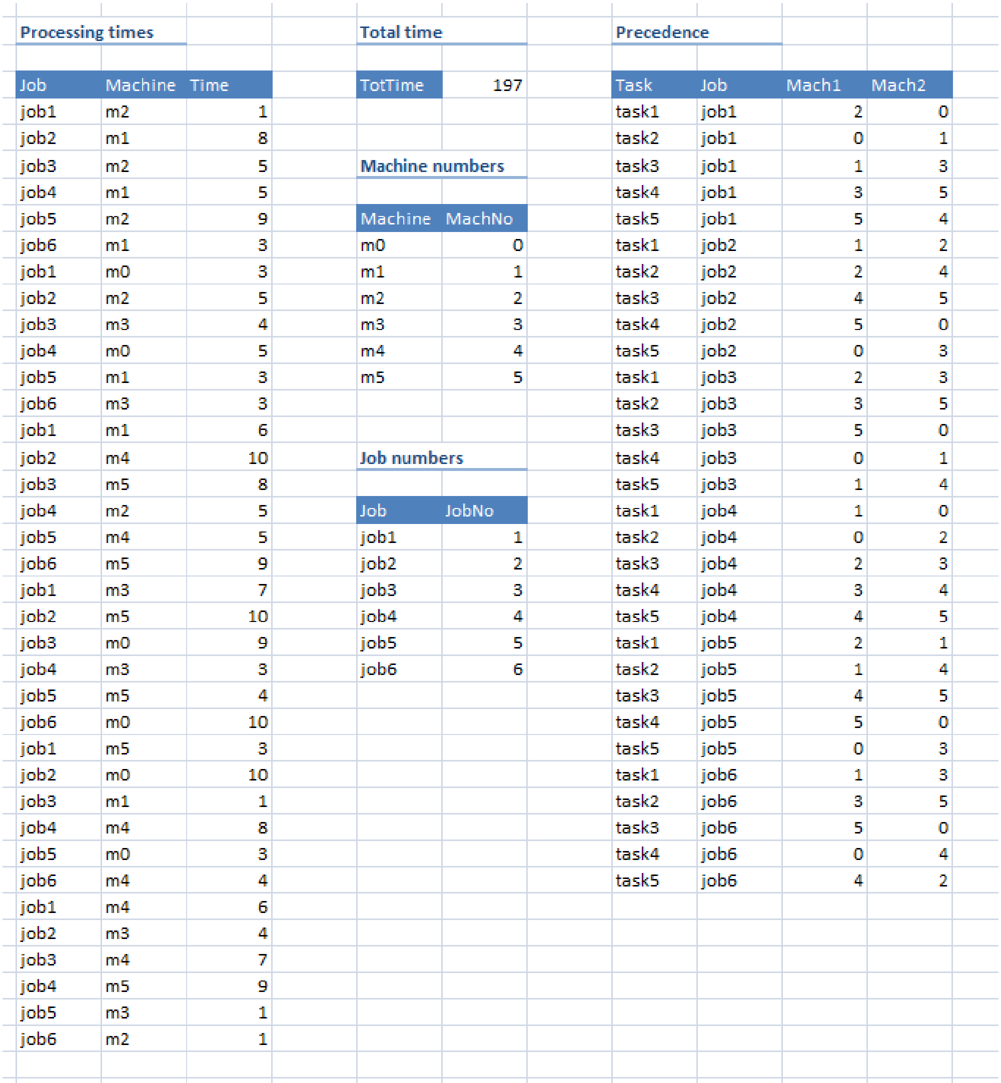
This formulation is from (Manne, 1960). The OML representation looks like:

```
// Job shop scheduling
Model[

  Parameters[Sets,Job,Machine,Task],
  Parameters[Reals,Time[Job, Machine],TotTime[]],
  Parameters[Integers,MachNo[Machine],Mach1[Task,Job],Mach2[Task,Job],JobNo[Job]],

  Decisions[Reals[0,Infinity],
         x[Job,Machine],  // start time of sub-task
         MakeSpan           // total make span of the problem
  ],
  // the 0-1 variables deal with overlap
  Decisions[Integers[0,1],y[Job,Job,Machine]],

  Constraints[

    // precedence
     Foreach[{t,Task},{j,Job},
         FilteredSum[{m2,Machine},MachNo[m2]==Mach2[t,j],x[j,m2]] >=
         FilteredSum[{m1,Machine},MachNo[m1]==Mach1[t,j],x[j,m1] + Time[j,m1]]
     ],

     // no overlap
     Foreach[{m,Machine},{j,Job},
        FilteredForeach[{k,Job},JobNo[j]<JobNo[k],
           x[j,m] >= x[k,m] + Time[k,m] - TotTime[]*y[j,k,m]
       ]],

     Foreach[{m,Machine},{j,Job},
        FilteredForeach[{k,Job},JobNo[j]<JobNo[k],
           x[k,m] >= x[j,m] + Time[j,m] - TotTime[]*(1-y[j,k,m])
       ]],

      // make span
      Foreach[{j,Job},{m,Machine},
         MakeSpan >= x[j,m] + Time[j,m]
      ]
  ],

   Goals[Minimize[makespan->MakeSpan]]

]
```

The precedence equations are somewhat complicated as we need to simulate a sparse set here:

$$x_{j,m2} \geq x_{j,m1} + t_{j,m1} \;\forall (j, m1, m2) \in S$$

The data as presented in the table is not suited to be imported directly into the model. In a different sheet we prepare the data ready for consumption by the model:



Processing times

Job	Machine	Time
job1	m2	1
job2	m1	8
job3	m2	5
job4	m1	5
job5	m2	9
job6	m1	3
job1	m0	3
job2	m2	5
job3	m3	4
job4	m0	5
job5	m1	3
job6	m3	3
job1	m1	6
job2	m4	10
job3	m5	8
job4	m2	5
job5	m4	5
job6	m5	9
job1	m3	7
job2	m5	10
job3	m0	9
job4	m3	3
job5	m5	4
job6	m0	10
job1	m5	3
job2	m0	10
job3	m1	1
job4	m4	8
job5	m0	3
job6	m4	4
job1	m4	6
job2	m3	4
job3	m4	7
job4	m5	9
job5	m3	1
job6	m2	1

Total time

TotTime	197

Machine numbers

Machine	MachNo
m0	0
m1	1
m2	2
m3	3
m4	4
m5	5

Job numbers

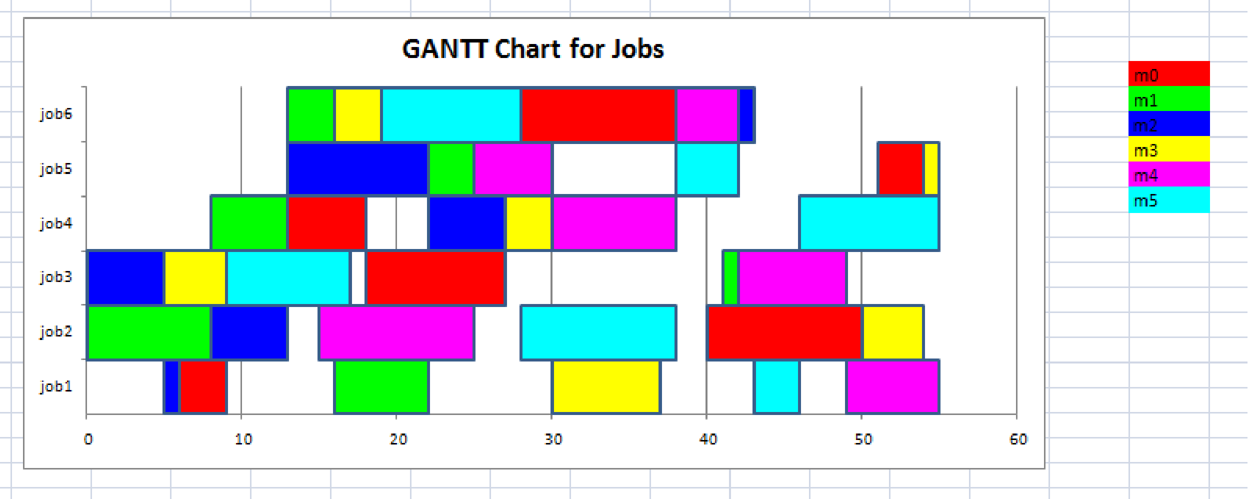
Job	JobNo
job1	1
job2	2
job3	3
job4	4
job5	5
job6	6

Precedence

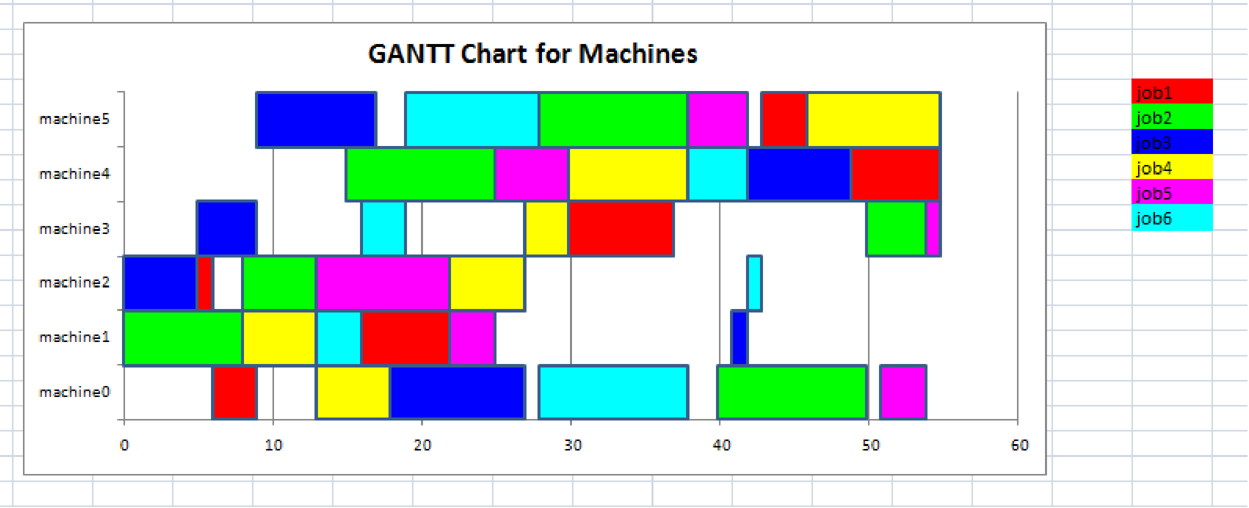
Task	Job	Mach1	Mach2
task1	job1	2	0
task2	job1	0	1
task3	job1	1	3
task4	job1	3	5
task5	job1	5	4
task1	job2	1	2
task2	job2	2	4
task3	job2	4	5
task4	job2	5	0
task5	job2	0	3
task1	job3	2	3
task2	job3	3	5
task3	job3	5	0
task4	job3	0	1
task5	job3	1	4
task1	job4	1	0
task2	job4	0	2
task3	job4	2	3
task4	job4	3	4
task5	job4	4	5
task1	job5	2	1
task2	job5	1	4
task3	job5	4	5
task4	job5	5	0
task5	job5	0	3
task1	job6	1	3
task2	job6	3	5
task3	job6	5	0
task4	job6	0	4
task5	job6	4	2

This problem with six jobs and six machines solves quickly with the Gurobi mip solver. The model can be simplified somewhat when using CSP modeling: the 'or' condition becomes easy and we can drop the binary *y* variables. However the CSP model did not solve as quickly as the MIP model.

With some VBA code it is easy to create a GANTT chart of the solution:



This shows how jobs are scheduled. A different view would be:

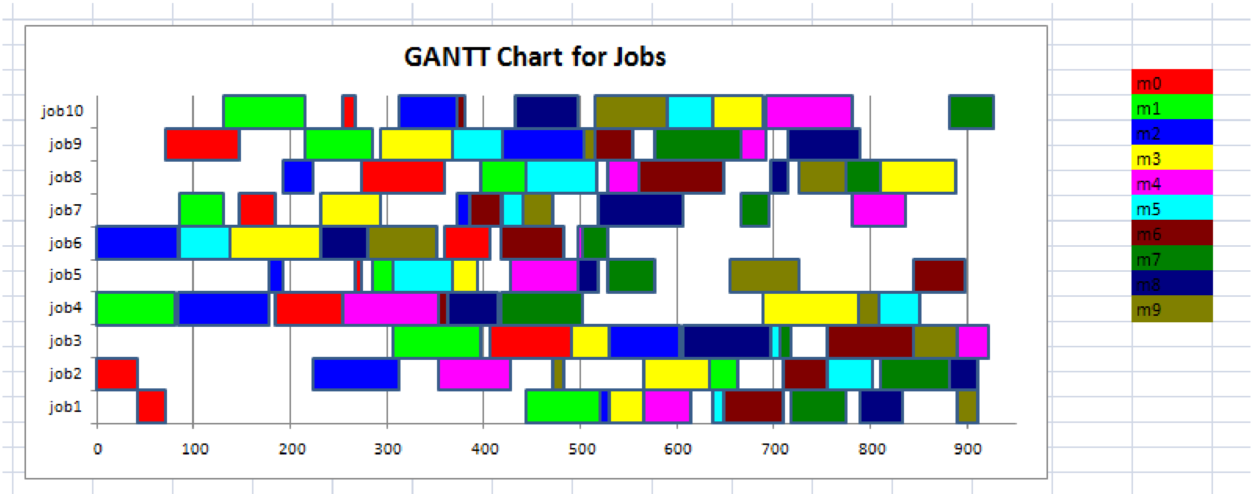


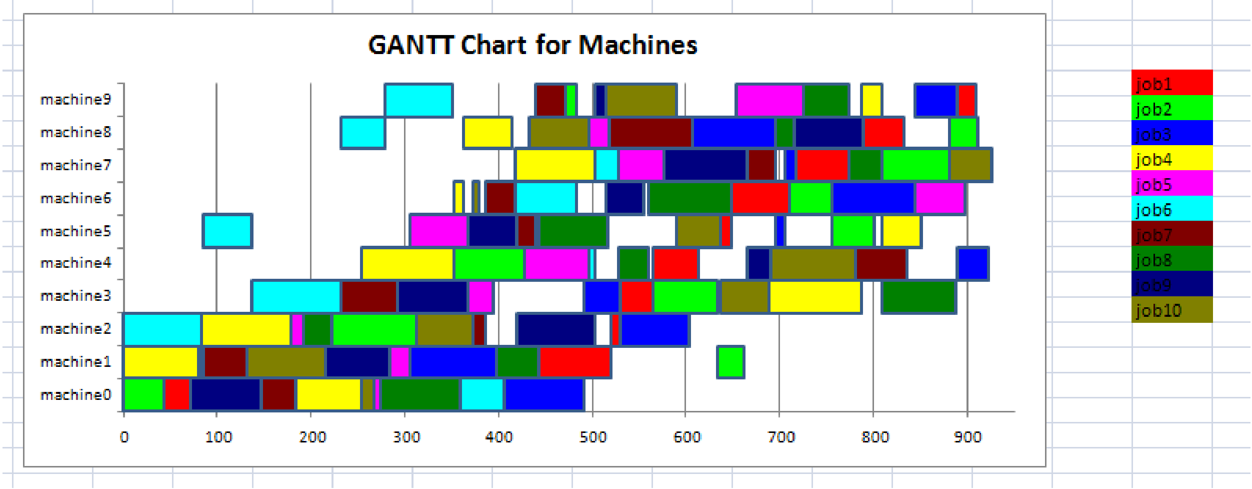
Larger instances can be difficult to solve. A famous benchmark model called ft10 (Fisher & Thompson, 1963) with 10 jobs and 10 machines was only solved to optimality in (Carlier & Pinson, 1989) after 25 years being unsolved (Jain & Meeran, 1998). Nowadays we can solve this problem using a MIP solver like Gurobi in less than five minutes on a standard PC.

The data for ft10 looks like:

	task1		task2		task3		task4		task5		task6		task7		task8		task9		task10	
	machine	time	machine	time	machine	time	machine	time	machine	time	machine	time	machine	time	machine	time	machine	time	machine	time
job1	m0	29	m1	78	m2	9	m3	36	m4	49	m5	11	m6	62	m7	56	m8	44	m9	21
job2	m0	43	m2	90	m4	75	m9	11	m3	69	m1	28	m6	46	m5	46	m7	72	m8	30
job3	m1	91	m0	85	m3	39	m2	74	m8	90	m5	10	m7	12	m6	89	m9	45	m4	33
job4	m1	81	m2	95	m0	71	m4	99	m6	9	m8	52	m7	85	m3	98	m9	22	m5	43
job5	m2	14	m0	6	m1	22	m5	61	m3	26	m4	69	m8	21	m7	49	m9	72	m6	53
job6	m2	84	m1	2	m5	52	m3	95	m8	48	m9	72	m0	47	m6	65	m4	6	m7	25
job7	m1	46	m0	37	m3	61	m2	13	m6	32	m5	21	m9	32	m8	89	m7	30	m4	55
job8	m2	31	m0	86	m1	46	m5	74	m4	32	m6	88	m8	19	m9	48	m7	36	m3	79
job9	m0	76	m1	69	m3	76	m5	51	m2	85	m9	11	m6	40	m7	89	m4	26	m8	74
job10	m1	85	m0	13	m2	61	m6	7	m8	64	m9	76	m5	47	m3	52	m4	90	m7	45

The results:





5.5 MAGIC SQUARES

A magic square is a square (n × n) matrix of unique integers 1,..,n² such that the row sums, column sums and sums over the two diagonals all yield the same number. For a detailed explanation see http://en.wikipedia.org/wiki/Magic_square.

The actual value of the sum over the rows, columns and diagonals can be easily derived. The sum over all cells is

$$\sum_{i=1}^{n^2} i = \frac{n^2}{2}(n^2+1)$$

Each row sum is equal, so a row sum is this number divided by *n*. As a result the row, column and diagonal sums are equal to

$$\frac{1}{2}n(n^2+1)$$

Finding a magic square of size *n* is a feasibility problem (there is no objective: just find a feasible solution). The problem is quite suited for the CSP solver, as we can use the all-different constraint to create a more efficient model than is possible with a MIP formulation. The basic model can look like:

```
Model[

  Parameters[Integers,N[]],
  Decisions[Integers[1,Infinity],Foreach[{i,1,N[]+1},{j,1,N[]+1},x[i,j]]],

  Constraints[

// bounds on variables:
     Foreach[{i,1,N[]+1},{j,1,N[]+1},1 <= x[i,j]<= N[]^2],

// row and column sums:
     Foreach[{i,1,N[]+1}, Sum[{j,1,N[]+1},x[i,j]] == N[]*(N[]^2+1)/2],
     Foreach[{j,1,N[]+1}, Sum[{i,1,N[]+1},x[i,j]] == N[]*(N[]^2+1)/2],

// diagonals:
     Sum[{i,1,N[]+1},x[i,i]] == N[]*(N[]^2+1)/2,
     Sum[{i,1,N[]+1},x[i,N[]-i+1]] == N[]*(N[]^2+1)/2,

// additional constraint: sum of all cells
     Sum[{i,1,N[]+1},{j,1,N[]+1},x[i,j]] == N[]^2*(N[]^2+1)/2,

// symmetry breaking constraints
     x[1,1] <= x[1,N[]]-1,
     x[1,1] <= x[N[],N[]]-1,
     x[1,1] <= x[N[],1]-1,
     x[1,N[]] <= x[N[],1]-1,

// all different
     Unequal[Foreach[{i,1,N[]+1},{j,1,N[]+1},x[i,j]]]
   ]

]
```

There are a number of points we can make about this model.

a. We repeat the expression `N[]*(N[]^2+1)/2` several times. That is really considered bad modeling. Unfortunately OML does not allow to say `Parameters[Integers,M=N[]*(N[]^2+1)/2]`. We could have calculated this number in Excel and import that, but that violates another rule: keep interfaces as small as possible.

b. We use here indexing 1..N. Although most modelers would prefer to use `X[1..n,1..n]`, OML models are often better readable when using `X[0..n-1,0..n-1]`. The reason is that `Sum[{i,N},..]` and `Foreach[{i,N},…]` actually means loop over i=0..N-1. A loop i=1..N can be written as `Sum[{i,1,N+1},..]` and `Foreach[{i,1,N+1},…]`. The visual clutter is exacerbated by using the `N[]` notation. My guess is that this zero-based indexing is chosen as MSF is really programmer-centric: it primarily targets and is developed by software developers as opposed to modelers.

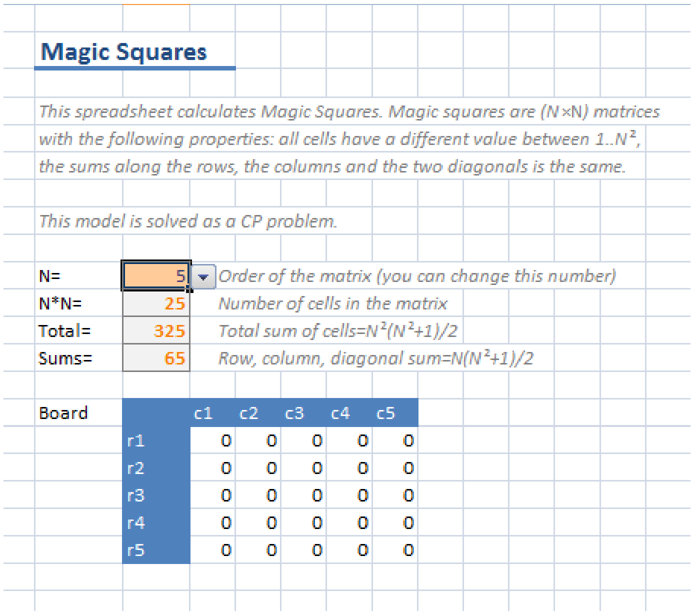
c. The all-different constraint is very useful for models like this. This construct cannot be translated efficiently to a MIP based model (Williams & Yan, 2001).

d. In MSF version 1.0, a bug did not allow us to declare a variable sized using N[]. I had to introduce a constant MaxN and size x accordingly. This resulted in unused variables.

e. In MSF version 1.0, the unused variables were printed in the solution sheet, in version 1.1, they are not. This caused problems with some of the spreadsheet applications I developed. It is more important to make new versions backward compatible if users build bigger application around a framework.

f. We use a few simple constraints to reduce some symmetry. For more examples of symmetry breaking constraints see section 5.9.

This application has a little GUI. The code for this has been implemented in Excel's VBA.



Magic Squares

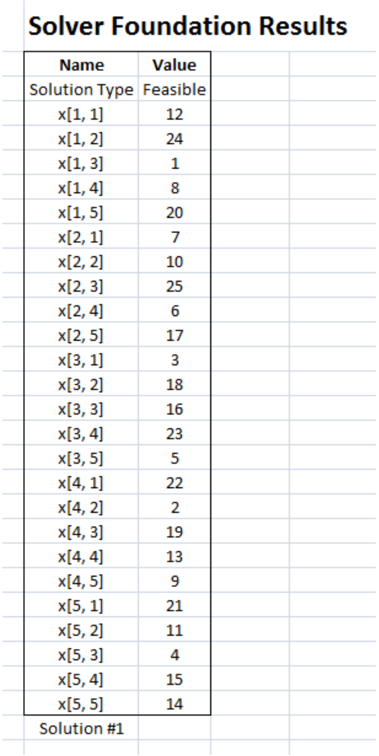
This spreadsheet calculates Magic Squares. Magic squares are (N×N) matrices with the following properties: all cells have a different value between 1..N^2, the sums along the rows, the columns and the two diagonals is the same.

This model is solved as a CP problem.

N=	5	*Order of the matrix (you can change this number)*
N*N=	25	*Number of cells in the matrix*
Total=	325	*Total sum of cells=$N^2(N^2+1)/2$*
Sums=	65	*Row, column, diagonal sum=$N(N^2+1)/2$*

Board	c1	c2	c3	c4	c5
r1	0	0	0	0	0
r2	0	0	0	0	0
r3	0	0	0	0	0
r4	0	0	0	0	0
r5	0	0	0	0	0

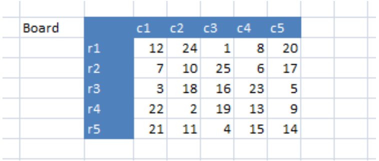
After setting N, the boards is resized to N×N, and the solution is reset to all zeros. After pressing the Solve button, MSF will solve the model:



Solver Foundation Results

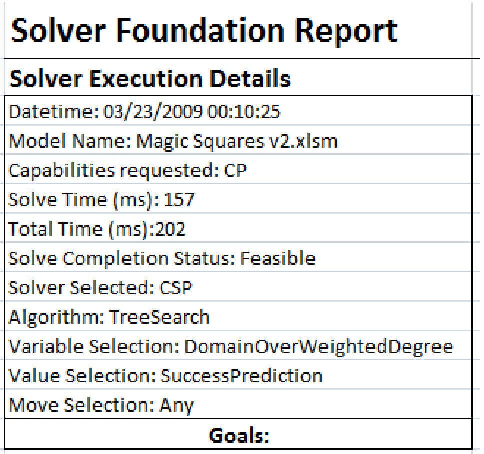
Name	Value
Solution Type	Feasible
x[1, 1]	12
x[1, 2]	24
x[1, 3]	1
x[1, 4]	8
x[1, 5]	20
x[2, 1]	7
x[2, 2]	10
x[2, 3]	25
x[2, 4]	6
x[2, 5]	17
x[3, 1]	3
x[3, 2]	18
x[3, 3]	16
x[3, 4]	23
x[3, 5]	5
x[4, 1]	22
x[4, 2]	2
x[4, 3]	19
x[4, 4]	13
x[4, 5]	9
x[5, 1]	21
x[5, 2]	11
x[5, 3]	4
x[5, 4]	15
x[5, 5]	14
Solution #1	

Finally when you go back to the main page, the result is displayed on the board:



Board		c1	c2	c3	c4	c5
	r1	12	24	1	8	20
	r2	7	10	25	6	17
	r3	3	18	16	23	5
	r4	22	2	19	13	9
	r5	21	11	4	15	14

The performance of this model depends on the algorithm settings. From the Summary we see that we used TreeSearch/DomainOverWeightedTree/SuccessPrediction. The default options cause the model to solve much slower.



Solver Foundation Report

Solver Execution Details

Datetime: 03/23/2009 00:10:25

Model Name: Magic Squares v2.xlsm

Capabilities requested: CP

Solve Time (ms): 157

Total Time (ms):202

Solve Completion Status: Feasible

Solver Selected: CSP

Algorithm: TreeSearch

Variable Selection: DomainOverWeightedDegree

Value Selection: SuccessPrediction

Move Selection: Any

Goals:

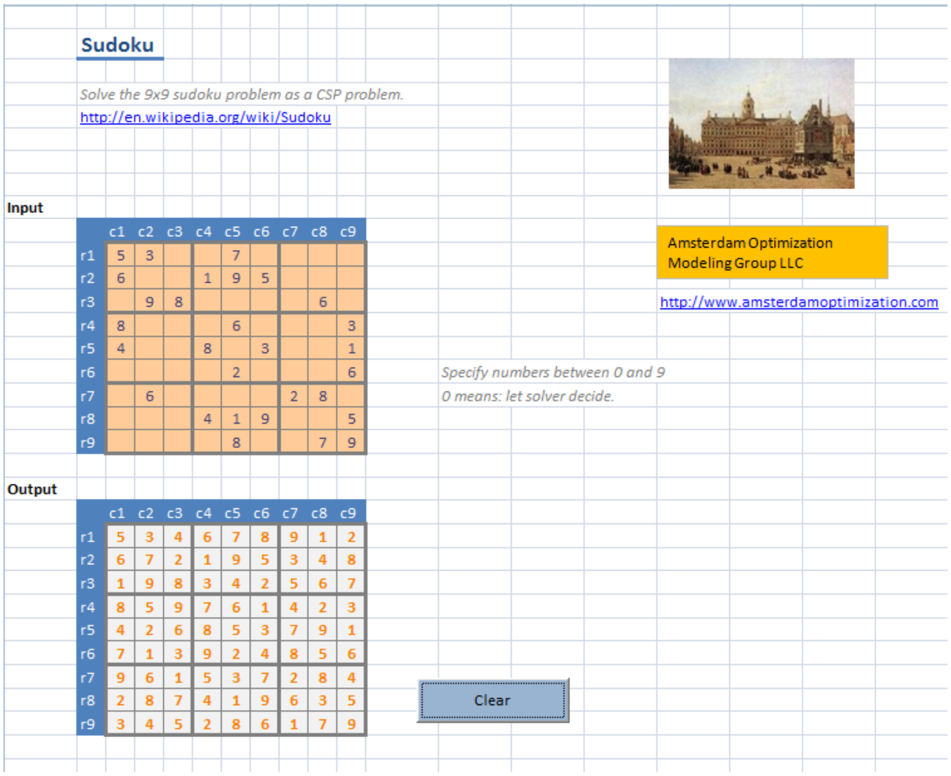
5.6 SUDOKU

The Sudoku puzzle (http://en.wikipedia.org/wiki/Sudoku) can be conveniently coded as a CSP problem.

```
Model[

  Parameters[Sets,I,J],
  Parameters[Integers,d[I,J]],

  Decisions[Integers[1,9],x[I,J]],

  Constraints[
    FilteredForeach[{i,I},{j,J},d[i,j]>0,x[i,j]==d[i,j]],
    Foreach[{i,I},Unequal[Foreach[{j,J},x[i,j]]]],
    Foreach[{j,J},Unequal[Foreach[{i,I},x[i,j]]]],
    Foreach[{ib,3},
        Foreach[{jb,3},
            Unequal[Foreach[{i,ib*3,ib*3+3},{j,jb*3,jb*3+3},x[i,j]]]
         ]
    ]
]

]
```

Notice how Foreach is placed both inside and outside the Unequal construct.

A small GUI is easily implemented in Excel:



Sudoku

Solve the 9x9 sudoku problem as a CSP problem.

http://en.wikipedia.org/wiki/Sudoku

Amsterdam Optimization Modeling Group LLC

http://www.amsterdamoptimization.com

Input

	c1	c2	c3	c4	c5	c6	c7	c8	c9
r1	5	3			7				
r2	6			1	9	5			
r3		9	8					6	
r4	8				6				3
r5	4			8		3			1
r6					2				6
r7		6					2	8	
r8				4	1	9			5
r9					8			7	9

Specify numbers between 0 and 9

0 means: let solver decide.

Output

	c1	c2	c3	c4	c5	c6	c7	c8	c9
r1	5	3	4	6	7	8	9	1	2
r2	6	7	2	1	9	5	3	4	8
r3	1	9	8	3	4	2	5	6	7
r4	8	5	9	7	6	1	4	2	3
r5	4	2	6	8	5	3	7	9	1
r6	7	1	3	9	2	4	8	5	6
r7	9	6	1	5	3	7	2	8	4
r8	2	8	7	4	1	9	6	3	5
r9	3	4	5	2	8	6	1	7	9

Checking whether the problem has a unique solution is easy: press the Next button. It should give: Final Solution Found.

5.7 LANGFORD SEQUENCES

Langford's problem is to form a sequence of numbers 1,…,n with some special characteristics. E.g. if we have 4 pairs of numbers denoted by L(2,4) then we can form:

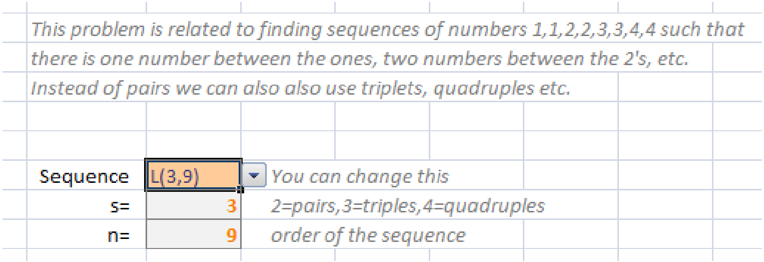
4	1	3	1	2	4	3	2

Between any pairs of value k there are k blocks with other numbers. For more information see: http://www.lclark.edu/~miller/langford.html. To model this we form variables Position[i,j] with *1 ≤ i ≤ n* the numbers to use and *1 ≤ j ≤ s* the number of duplicates (j=2 means pairs). The constraints are simple: the position should be between 1 and n*s and should be unique. This can be easily modeled using the Unequal construct. Secondly we impose that *x[i,j] - x[i,j-1] = i+1*. The complete model is:

```
Model[

  Parameters[Integers,N[],S[] ],

  Decisions[Integers,Foreach[{i,1,N[]+1},{j,1,S[]+1},position[i,j]]],

 Constraints[
   Foreach[{i,1,N[]+1},{j,1,S[]+1},  1 <= position[i,j] <= S[]*N[]],

   Foreach[{i,1,N[]+1},{j,2,S[]+1},  position[i,j] == position[i,j-1] + i + 1 ],
   Unequal[Foreach[{i,1,N[]+1},{j,1,S[]+1},position[i,j]]]
  ]

]
```

For a MIP formulation see: http://www.amsterdamoptimization.com/pdf/langford.pdf. There is more research available about CSP modeling with respect to this problem (Smith, 2000).

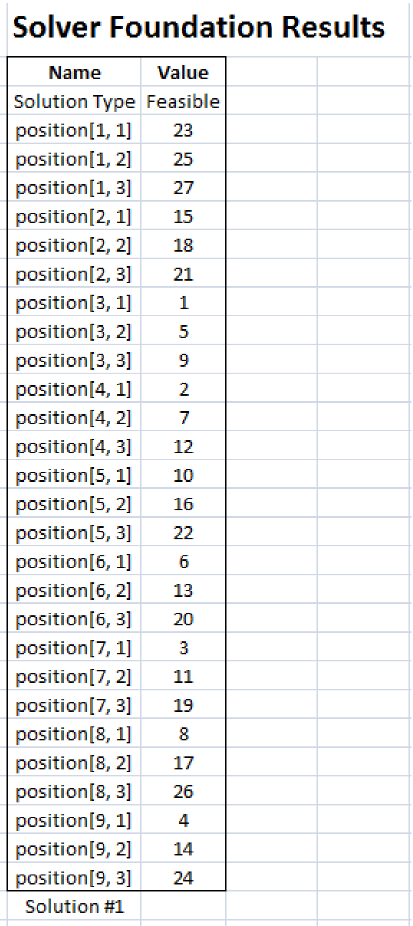
The GUI allows you to select from a number of possible sequences:



This problem is related to finding sequences of numbers 1,1,2,2,3,3,4,4 such that there is one number between the ones, two numbers between the 2's, etc. Instead of pairs we can also also use triplets, quadruples etc.

Sequence	L(3,9)	*You can change this*
s=	3	*2=pairs,3=triples,4=quadruples*
n=	9	*order of the sequence*

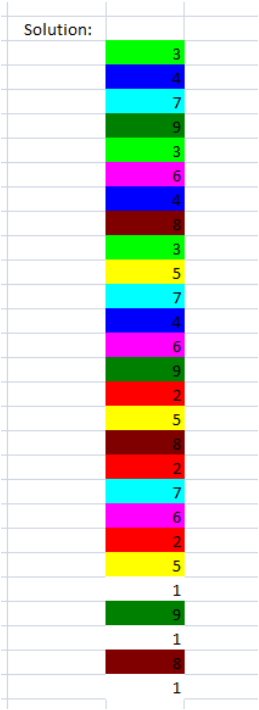
After selecting the problem, you can press the Solve button. The reported solution looks like:



Solver Foundation Results

Name	Value
Solution Type	Feasible
position[1, 1]	23
position[1, 2]	25
position[1, 3]	27
position[2, 1]	15
position[2, 2]	18
position[2, 3]	21
position[3, 1]	1
position[3, 2]	5
position[3, 3]	9
position[4, 1]	2
position[4, 2]	7
position[4, 3]	12
position[5, 1]	10
position[5, 2]	16
position[5, 3]	22
position[6, 1]	6
position[6, 2]	13
position[6, 3]	20
position[7, 1]	3
position[7, 2]	11
position[7, 3]	19
position[8, 1]	8
position[8, 2]	17
position[8, 3]	26
position[9, 1]	4
position[9, 2]	14
position[9, 3]	24
Solution #1	

After going back to the main sheet, the solution is presented in a different, graphical format:



Solution:	
	3
	4
	7
	9
	3
	6
	4
	8
	3
	5
	7
	4
	6
	9
	2
	5
	8
	2
	7
	6
	2
	5
	1
	9
	1
	8
	1

VBA code was used to form this solution.

5.8 TRAVELING SALESMAN PROBLEM

In theory it is easy to formulate a TSP when the all-different constraint is present. Let `x[i]` be the ith city visited, then an OML-like formulation could look like:

```
Model[
  Parameters[Integers,N=14],
  Parameters[Sets[Integers],city],
  Parameters[Reals,dist[city,city]],
  Decisions[Integers[0,N-1],x[city]],
  Constraints[
     Unequal[Foreach[{i,city},x[i]]]
  ],
  Goals[
    Minimize[Sum[{i,city},dist[x[i],x[i++1]]]]
  ]
]
```

This does not work completely. First, bounds cannot contain a symbolic constant. So we need to write `[0,13]` instead of `[0,N-1]` as bounds: `Decisions[Integers[0,13],x[city]]`. The objective has a few more difficulties. There is no circular lead/lag operator in OML (like the `--` or `++` operator in GAMS). This can be solved by introducing a parameter `next[city]`. This can be calculated in Excel and imported in the OML model. Furthermore, we cannot use a variable as an index, so `dist[x[i],x[next[i]]]` is not a valid construct (some languages geared towards solving CSP problems allow this; it adds concise expressiveness to the language that may help the modeler). The error message indicates this is not related to OML per se but rather to solver support. Also it is not allowed to use something like `FilteredSum[{j,city},j==x[i],...]` (a condition in a FilteredSum cannot depend on a decision variable). The only thing I could think of was:

```
Minimize[Sum[{i,city},{j,city},{k,city},AsInt[j==x[i]]*AsInt[k==x[next[i]]]*dist[j,k]]]
```

Note that I used integers as set elements. I started with using data-binding through tables, using set elements *{'city0','city1',...,'city13'}*. This made the model somewhat more complicated, as x[i] is an integer. So the CSP formulation uses the simpler set *{0,1,...,13}* allowing us to operate on indices. (Some consider this bad modeling: in GAMS arithmetic on sets is actually discouraged, and needs a function `ord()` to convert the element — always a string in GAMS — to an integer). Unfortunately I was not able to solve the small 14 city example with the CSP solver using this formulation. (Even after adding `City[0]==0` as constraint). To check the input, I also tried a MIP formulation with Gurobi. That solved this small instance just fine.

In the MIP formulation we needed to formulate:

```
FilteredSum[{i,city},i != 'city0', x['city0',i]]==1
```

From what I can see this is not possible: a set element not being an integer can not be used in OML. As a workaround I created sets by binding to some dummy parameters:

```
Sum[{i0,city0},{i,city2},x[i0,i]]==1
```

where `city0` is a set containing only a single element *'city0'*, and `city2` is a set *{'city1',...,'city13'}*. The constraint

```
FilteredSum[{i,city},{j,city},i != j,x[i,j]] <= N
```

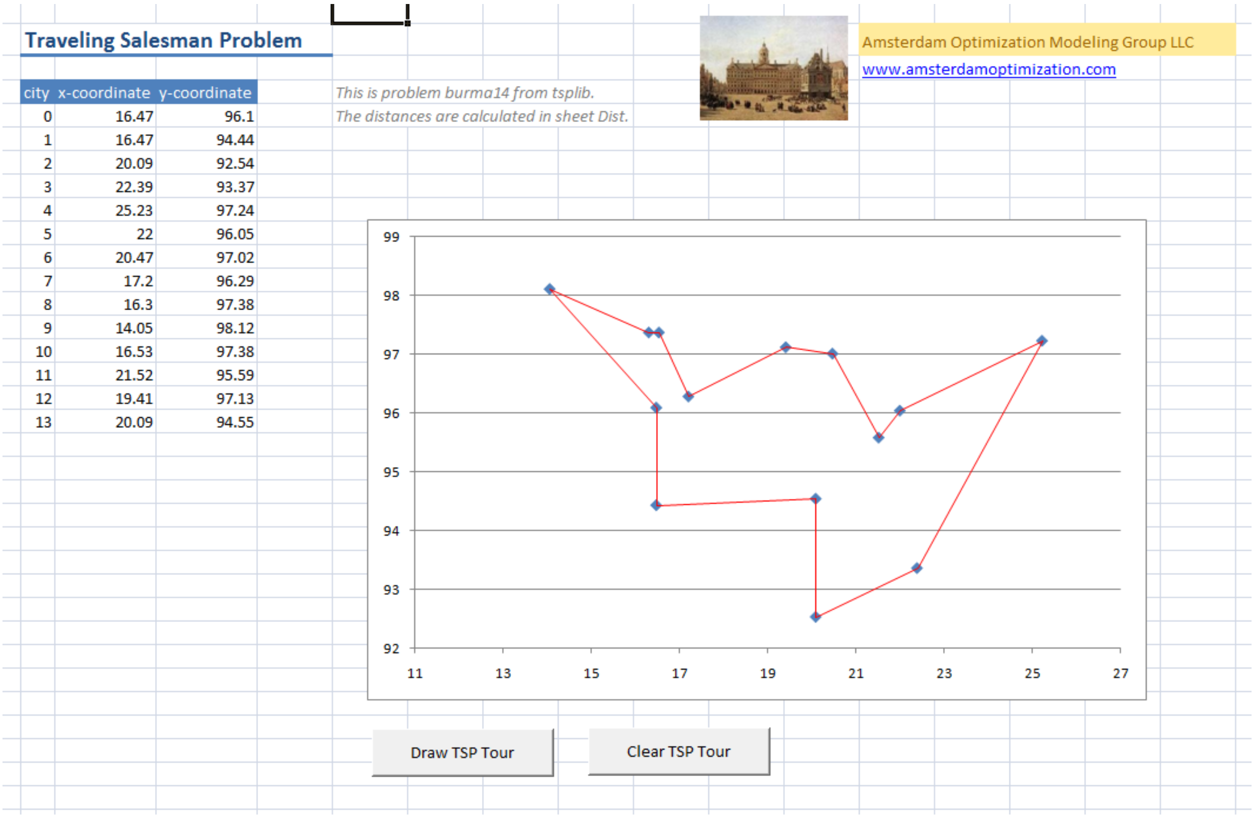
did not work as i and j are not numeric. A workaround could be to have a parameter `num[city]` which can be calculated in Excel and then imported. Then the condition can read: `num[i]!=num[j]`. I just used `dist[i,j]>0` as condition, as that also can be used to exclude the diagonal (the Excel spreadsheet makes sure all diagonal distances `dist[i,i]` are zero). A more general approach would be to be able to introduce sparse 2-d sets (like one could do in AMPL and GAMS), but OML has only one dimensional sets as far as I know.

This exercise was really meant to explore the expressiveness of OML. The little model actually shows some interesting issues with OML and some possible workarounds. I do not want to suggest this is a good way to solve TSP's. Obviously these approaches are not suited for large problems. A cutting plane algorithm often is quite effective for slightly larger problems (see this 42 city problem). The Excel plug-in does not allow us to implement such an algorithm: an OML model can only contain a single model and has no looping facilities. For the real large problems, you need to look elsewhere, such as Concorde.

The complete MIP model looks like:

```
Model[

// problem burma14 from tsplib

  Parameters[Integers,N=14],
  Parameters[Reals,f=0.1],
  Parameters[Sets,city,city2,city0],
  Parameters[Integers,dist[city,city]],
  Parameters[Integers,dummy[city2]],
  Parameters[Integers,dummy2[city0]],


  Decisions[Integers[0,1],x[city,city]],
  Decisions[Reals[0,Infinity],y[city,city]],

  Constraints[

     // Svestka formulation
     Foreach[{i,city2}, FilteredSum[{j,city},dist[i,j]>0, y[j,i]] >= 1],
     Foreach[{i,city2}, FilteredSum[{j,city},dist[i,j]>0,y[i,j]] - FilteredSum[{j,city},dist[i,j]>0,
y[j,i]] == f],
     FilteredSum[{i,city},{j,city}, dist[i,j]>0,x[i,j]] <= N,
     FilteredForeach[{i,city},{j,city},dist[i,j]>0, y[i,j] <= (1 + N*f)*x[i,j]],

     // tighten model
     Foreach[{i,city2},FilteredSum[{j,city},dist[i,j]>0, x[j,i]] == 1],
     Foreach[{i,city2},FilteredSum[{j,city},dist[i,j]>0,x[i,j]] == FilteredSum[{j,city},dist[i,j]>0,
x[j,i]]],
     Sum[{i0,city0},{i,city2},x[i0,i]]==1,
     Sum[{i0,city0},{i,city2},y[i0,i]]==1
   ],


  Goals[
  Minimize[FilteredSum[{i,city},{j,city}, dist[i,j]>0,dist[i,j]*x[i,j]]]
  ]


]
```

This formulation is from (Svestka, 1978). The problem is from TSPLIB and the distance calculations are replicated in the Excel spreadsheet (they involve complicated expressions due to projections). The final result is displayed in a chart:



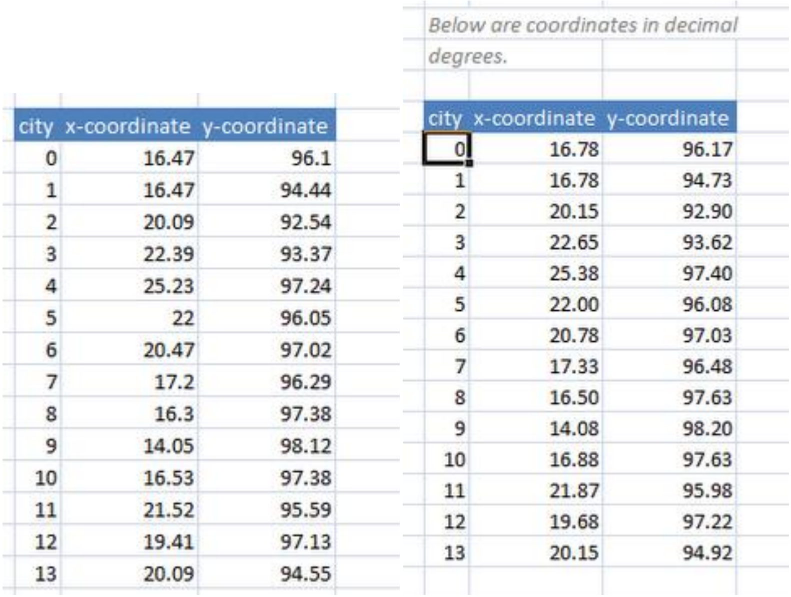
Traveling Salesman Problem

city	x-coordinate	y-coordinate
0	16.47	96.1
1	16.47	94.44
2	20.09	92.54
3	22.39	93.37
4	25.23	97.24
5	22	96.05
6	20.47	97.02
7	17.2	96.29
8	16.3	97.38
9	14.05	98.12
10	16.53	97.38
11	21.52	95.59
12	19.41	97.13
13	20.09	94.55

This is problem burma14 from tsplib.
The distances are calculated in sheet Dist.

Amsterdam Optimization Modeling Group LLC
www.amsterdamoptimization.com

A slightly better picture results if we use decimal degrees. The coordinates above are in the format DD.MM (degrees and minutes). To convert this to decimal degrees we use the formula: DD+MM/60. The result is show below:

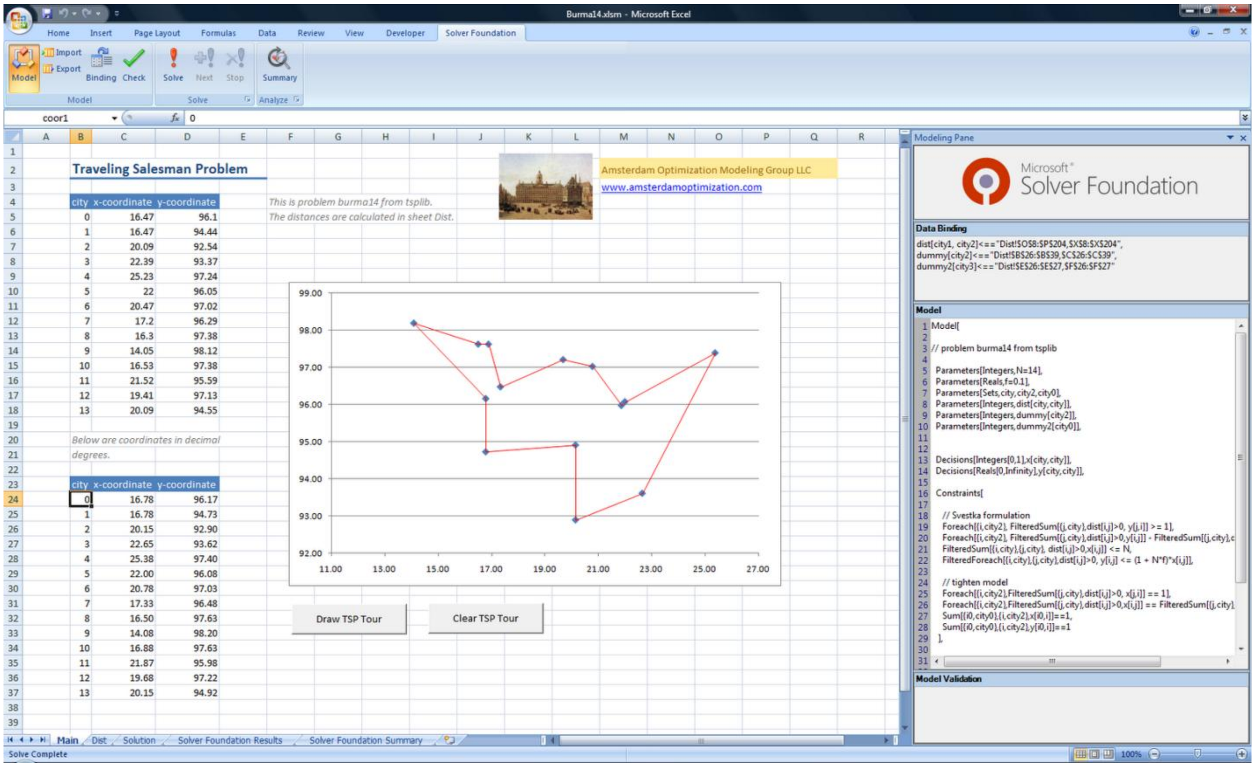


city	x-coordinate	y-coordinate
0	16.47	96.1
1	16.47	94.44
2	20.09	92.54
3	22.39	93.37
4	25.23	97.24
5	22	96.05
6	20.47	97.02
7	17.2	96.29
8	16.3	97.38
9	14.05	98.12
10	16.53	97.38
11	21.52	95.59
12	19.41	97.13
13	20.09	94.55

Below are coordinates in decimal degrees.

city	x-coordinate	y-coordinate
0	16.78	96.17
1	16.78	94.73
2	20.15	92.90
3	22.65	93.62
4	25.38	97.40
5	22.00	96.08
6	20.78	97.03
7	17.33	96.48
8	16.50	97.63
9	14.08	98.20
10	16.88	97.63
11	21.87	95.98
12	19.68	97.22
13	20.15	94.92

The differences are small but noticeable. The optimal tour stays the same as the calculation of the distance matrix did not change as a result of this.



Traveling Salesman Problem

city	x-coordinate	y-coordinate
0	16.47	96.1
1	16.47	94.44
2	20.09	92.54
3	22.39	93.37
4	25.23	97.24
5	22	96.05
6	20.47	97.02
7	17.2	96.29
8	16.3	97.38
9	14.05	98.12
10	16.53	97.38
11	21.52	95.59
12	19.41	97.13
13	20.09	94.55

This is problem burma14 from tsplib.
The distances are calculated in sheet Dist.

Amsterdam Optimization Modeling Group LLC
www.amsterdamoptimization.com

Below are coordinates in decimal degrees.

city	x-coordinate	y-coordinate
0	16.78	96.17
1	16.78	94.73
2	20.15	92.90
3	22.65	93.62
4	25.38	97.40
5	22.00	96.08
6	20.78	97.03
7	17.33	96.48
8	16.50	97.63
9	14.08	98.20
10	16.88	97.63
11	21.87	95.98
12	19.68	97.22
13	20.15	94.92

Data Binding

```
dist[city1, city2]<=="Dist!$O$8:$P$204,$X$8:$X$204",
dummy[city2]<=="Dist!$B$26:$B$39,$C$26:$C$39",
dummy2[city3]<=="Dist!$E$26:$E$27,$F$26:$F$27"
```

Model

```
Model[

// problem burma14 from tsplib

 Parameters[Integers,N=14],
 Parameters[Reals,f=0.1],
 Parameters[Sets,city,city2,city0],
 Parameters[Integers,dist[city,city]],
 Parameters[Integers,dummy[city2]],
 Parameters[Integers,dummy2[city0]],


 Decisions[Integers[0,1],x[city,city]],
 Decisions[Reals[0,Infinity],y[city,city]],

 Constraints[

   // Svestka formulation
   Foreach[{i,city2}, FilteredSum[{j,city},dist[i,j]>0, y[j,i]] >= 1],
   Foreach[{i,city2}, FilteredSum[{j,city},dist[i,j]>0,y[i,j]] - FilteredSum[{j,city},c
   FilteredSum[{i,city},{j,city}, dist[i,j]>0,x[i,j]] <= N,
   FilteredForeach[{i,city},{j,city},dist[i,j]>0, y[i,j] <= (1 + N*f)*x[i,j]],

   // tighten model
   Foreach[{i,city2},FilteredSum[{j,city},dist[i,j]>0, x[j,i]] == 1],
   Foreach[{i,city2},FilteredSum[{j,city},dist[i,j]>0,x[i,j]] == FilteredSum[{j,city},
   Sum[{i0,city0},{i,city2},x[i0,i]]==1,
   Sum[{i0,city0},{i,city2},y[i0,i]]==1
 ],
```

Model Validation

5.9 TILING SQUARES

The tiling problem is not very amenable for solving by MIP solvers (see tiling.pdf). In this problem we try to cover an (N × N) square with smaller squares. In a MIP we need big-M formulations to express the non-overlap condition. With CSP we can write this down more directly and conveniently. Instead of using something like:

$$
\begin{gathered}
x_i \geq x_j + s_j - M\delta_{1,i,j} \\
x_i + s_i \leq x_j + M\delta_{2,i,j} \\
y_i \geq y_j + s_j - M\delta_{3,i,j} \\
y_i + s_i \leq y_j + M\delta_{4,i,j} \\
\sum_k \delta_{k,i,j} \leq 3
\end{gathered}
$$

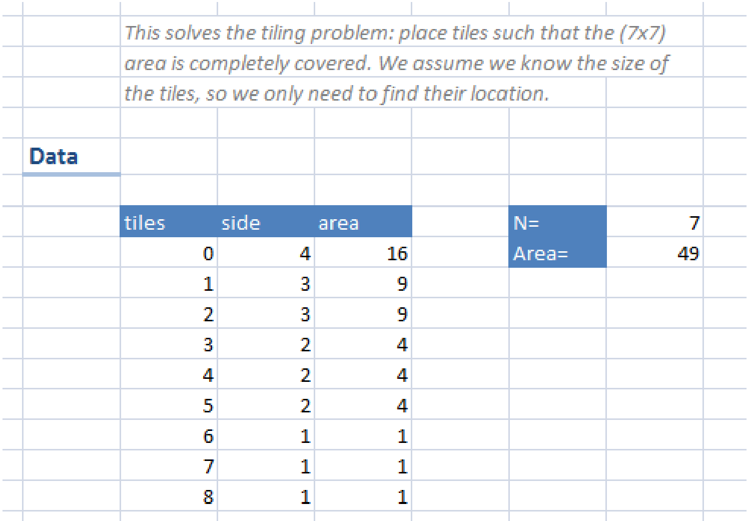
we can write in OML:

```
Foreach[{i,T},{j,i+1,T},
    x[i]>=x[j]+Side[j] |
    x[i]+Side[i]<=x[j] |
    y[i]>=y[j]+Side[j] |
    y[i]+Side[i]<=y[j]
]
```

Here the vertical bar means OR. A small (7 × 7) problem can be implemented quickly as follows:

```
Model[
    Parameters[Integers,N=7],   // size of square to fill
    Parameters[Integers,T=9],   // number of tiles

    Parameters[Sets,Tiles],
    Parameters[Integers,Side[Tiles]],

    Decisions[Integers[0,6],x[Tiles],y[Tiles]],

    Constraints[

         Foreach[{i,Tiles},x[i]<=N-Side[i] & y[i]<=N-Side[i]],

         Foreach[{i,T},{j,i+1,T},
             x[i]>=x[j]+Side[j]  |
             x[i]+Side[i]<=x[j]  |
             y[i]>=y[j]+Side[j]  |
             y[i]+Side[i]<=y[j]
         ]
    ]

]
```

Using this data:



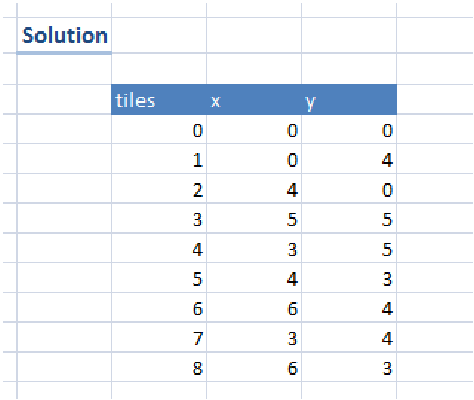
This solves the tiling problem: place tiles such that the (7x7) area is completely covered. We assume we know the size of the tiles, so we only need to find their location.

Data

tiles	side	area
0	4	16
1	3	9
2	3	9
3	2	4
4	2	4
5	2	4
6	1	1
7	1	1
8	1	1

N=	7
Area=	49

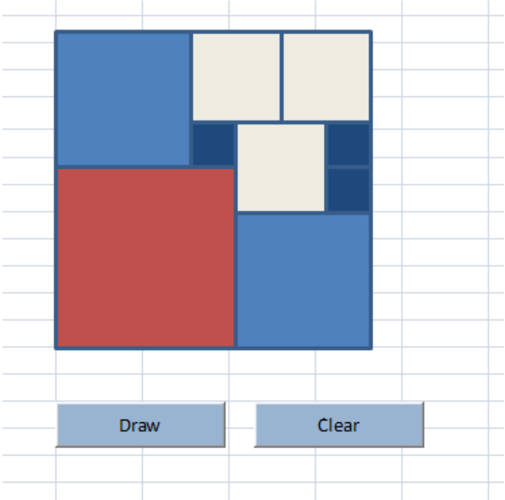
we get the solution:



Solution

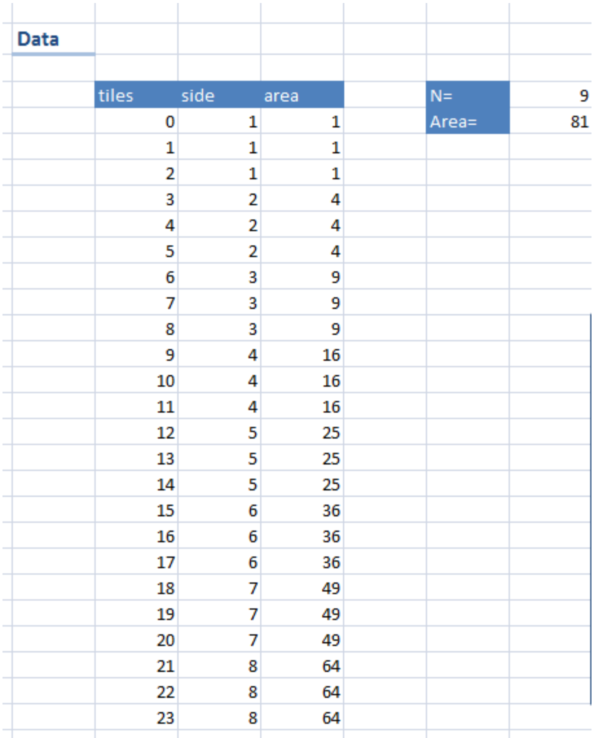
tiles	x	y
0	0	0
1	0	4
2	4	0
3	5	5
4	3	5
5	4	3
6	6	4
7	3	4
8	6	3

Using Excel we can display this a little bit more interesting:



Here we already used knowledge about which tiles to use. We only needed to determine where to place them. The real problem is slightly more complicated: we don't know which tiles to use in advance. Below, we are looking for a configuration for the (9 x 9) problem where no tile can be used more than three times. This is number *h(9)* in Integer Square Tilings and *a(9)* in sequence A036444. To formulate this problem we add boolean variables *b(i)* indicating if tile *i* is being used.

The data-set is now larger: all possible tiles smaller than the size of the area to tile are listed:



Data

tiles	side	area
0	1	1
1	1	1
2	1	1
3	2	4
4	2	4
5	2	4
6	3	9
7	3	9
8	3	9
9	4	16
10	4	16
11	4	16
12	5	25
13	5	25
14	5	25
15	6	36
16	6	36
17	6	36
18	7	49
19	7	49
20	7	49
21	8	64
22	8	64
23	8	64

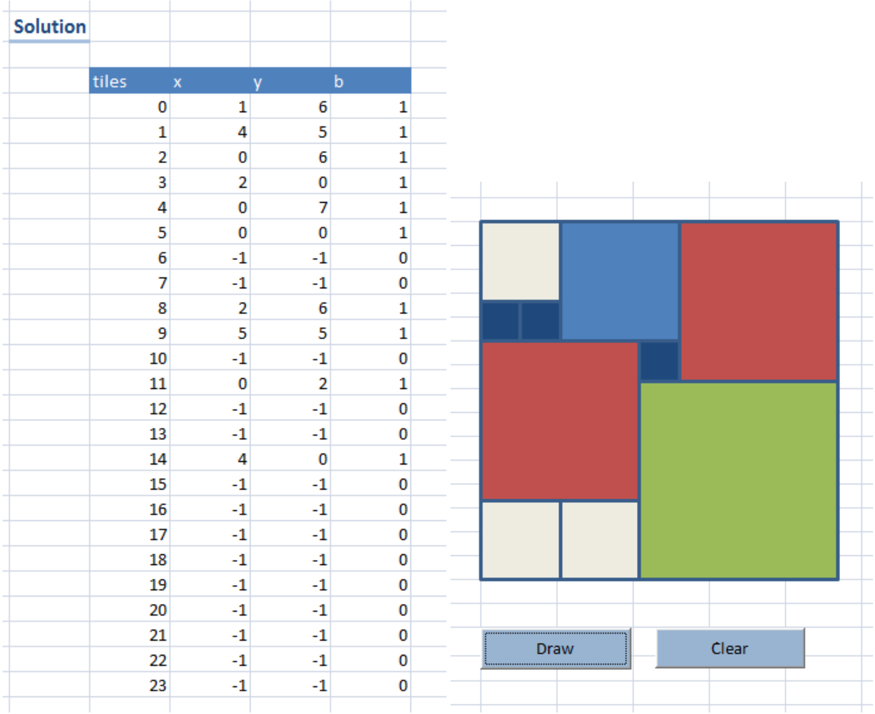
N=	9
Area=	81

The model for this version is:

```
Model[
    Parameters[Integers,N=9],   // size of square to fill
    Parameters[Integers,T=24],   // number of tiles

    Parameters[Sets,Tiles],
    Parameters[Integers,Side[Tiles]],

    Decisions[Booleans,b[Tiles]],
    Decisions[Integers[0,8],x[Tiles],y[Tiles]],



    Constraints[

         Foreach[{i,Tiles},Implies[b[i],x[i]<=N-Side[i] & y[i]<=N-Side[i]]],

         Foreach[{i,T},{j,i+1,T},Implies[b[i]&b[j],
             x[i]>=x[j]+Side[j]  |
             x[i]+Side[i]<=x[j]  |
             y[i]>=y[j]+Side[j]  |
             y[i]+Side[i]<=y[j]
         ]],

        Sum[{i,Tiles},Side[i]^2 * AsInt[b[i]]] == N^2
    ]

]
```

The Implies constructs implement an implication: if (condition) then some_constraint.

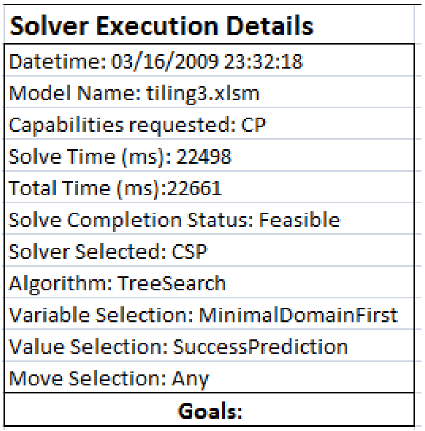
A solution for this problem is as follows:



Solution

tiles	x	y	b
0	1	6	1
1	4	5	1
2	0	6	1
3	2	0	1
4	0	7	1
5	0	0	1
6	-1	-1	0
7	-1	-1	0
8	2	6	1
9	5	5	1
10	-1	-1	0
11	0	2	1
12	-1	-1	0
13	-1	-1	0
14	4	0	1
15	-1	-1	0
16	-1	-1	0
17	-1	-1	0
18	-1	-1	0
19	-1	-1	0
20	-1	-1	0
21	-1	-1	0
22	-1	-1	0
23	-1	-1	0

The performance is somewhat mediocre:



Solver Execution Details

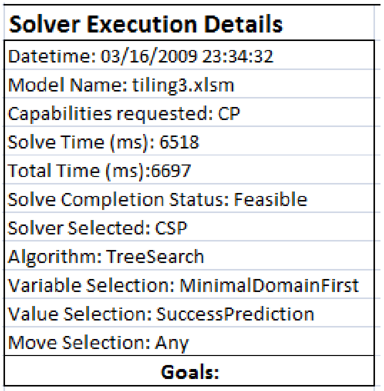
Datetime: 03/16/2009 23:32:18
Model Name: tiling3.xlsm
Capabilities requested: CP
Solve Time (ms): 22498
Total Time (ms):22661
Solve Completion Status: Feasible
Solver Selected: CSP
Algorithm: TreeSearch
Variable Selection: MinimalDomainFirst
Value Selection: SuccessPrediction
Move Selection: Any

Goals:

However we can improve this by adding a symmetry-breaking constraint that says we order identical tiles in the x-direction:

```
// this is to reduce some symmetry
Foreach[{i,T},{j,i+1,T},Implies[b[i]&b[j]&(Side[i]==Side[j]),x[j]>=x[i]]]
```

This reduces the solution time significantly:



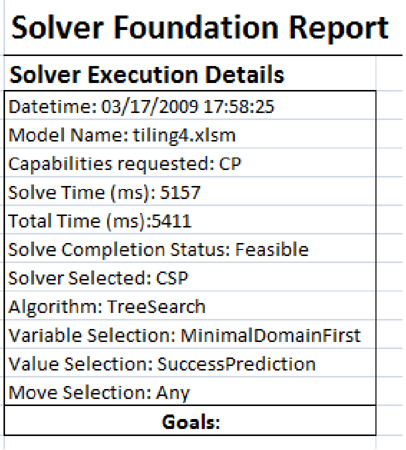
Solver Execution Details

Datetime: 03/16/2009 23:34:32
Model Name: tiling3.xlsm
Capabilities requested: CP
Solve Time (ms): 6518
Total Time (ms):6697
Solve Completion Status: Feasible
Solver Selected: CSP
Algorithm: TreeSearch
Variable Selection: MinimalDomainFirst
Value Selection: SuccessPrediction
Move Selection: Any
Goals:

A small improvement is possible by ordering tiles also in the y-direction (if equal x-position). As there are few cases where the x-position is identical we expect this just to be a small improvement, and indeed we see if we use:

```
// this is to reduce some symmetry
Foreach[{i,T},{j,i+1,T},Implies[b[i]&b[j]&(Side[i]==Side[j]),x[j]>=x[i]]],
Foreach[{i,T},{j,i+1,T},Implies[
            b[i]&b[j]&(Side[i]==Side[j])&(x[j]==x[i]),
            y[j]>=y[i]]]
```

that the timings are slightly better:



Solver Foundation Report

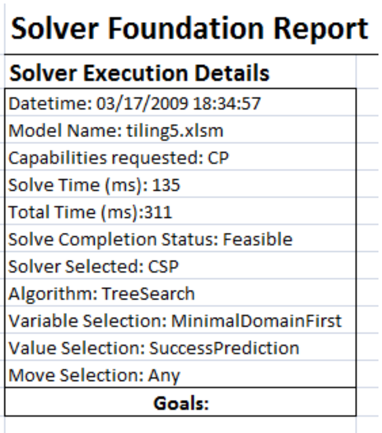
Solver Execution Details

Datetime: 03/17/2009 17:58:25
Model Name: tiling4.xlsm
Capabilities requested: CP
Solve Time (ms): 5157
Total Time (ms):5411
Solve Completion Status: Feasible
Solver Selected: CSP
Algorithm: TreeSearch
Variable Selection: MinimalDomainFirst
Value Selection: SuccessPrediction
Move Selection: Any
Goals:

A more pronounced improvement can be achieved by ordering the tiles when we place them. If we look again at the data we see that tiles 9, 10 and 11 all have size four. If we place one tile of size four we want to make sure it is 9. We can implement this by the constraint:

```
FilteredForeach[{i,1,T},Side[i]==Side[i-1],Implies[b[i],b[i-1]]]
```

This gives a big improvement:



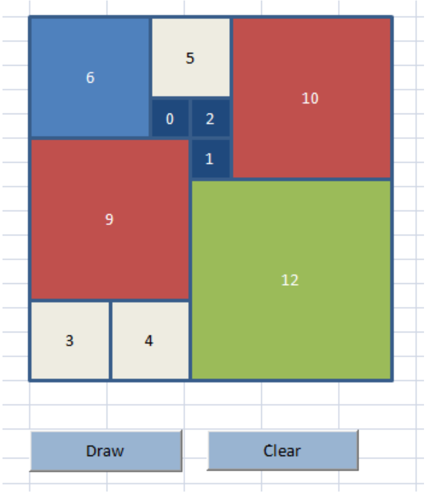
Solver Foundation Report

Solver Execution Details

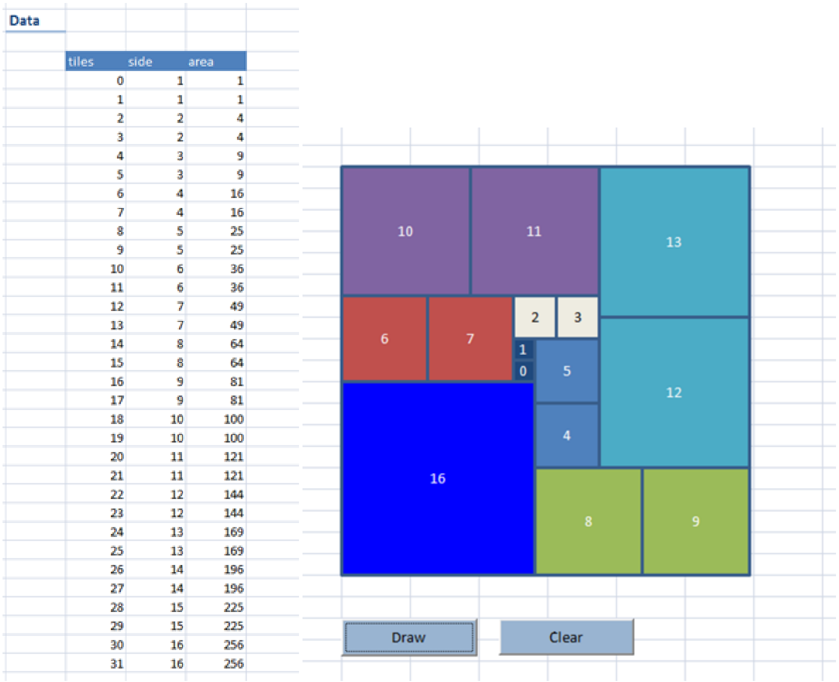
Datetime: 03/17/2009 18:34:57
Model Name: tiling5.xlsm
Capabilities requested: CP
Solve Time (ms): 135
Total Time (ms):311
Solve Completion Status: Feasible
Solver Selected: CSP
Algorithm: TreeSearch
Variable Selection: MinimalDomainFirst
Value Selection: SuccessPrediction
Move Selection: Any

Goals:

To make it easier to see which tiles are placed I added the tile number in the picture:



This now has good enough performance that we can try some big ones. The reference sequence A036444 shows *a(19)=2*. I.e. we can use just a set with 2 identical tiles to cover an area of (19 × 19). Indeed we can verify this with this model. The data and results are:



Data

tiles	side	area
0	1	1
1	1	1
2	2	4
3	2	4
4	3	9
5	3	9
6	4	16
7	4	16
8	5	25
9	5	25
10	6	36
11	6	36
12	7	49
13	7	49
14	8	64
15	8	64
16	9	81
17	9	81
18	10	100
19	10	100
20	11	121
21	11	121
22	12	144
23	12	144
24	13	169
25	13	169
26	14	196
27	14	196
28	15	225
29	15	225
30	16	256
31	16	256

5.10 ALPHAMETICS

The problem is to find a mapping between letters and digits (0,…,9) such that the addition is correct. A famous example is:

```
   S E N D
   M O R E
 ---------
 M O N E Y
```

This can be translated to:

```
   9 5 6 7
   1 0 8 5
 ---------
 1 0 6 5 2
```

A simple CSP model for this is:

```
Model[
   Decisions[Integers[0,9],s,e,n,d,m,o,r,y],
   Constraints[
     1e3*s + 100*e + 10*n + d + 1000*m + 100*o + 10*r + e ==
       10000*m + 1000*o + 100*n + 10*e + y,
     Unequal[s,e,n,d,m,o,r,y],
     s>=1,m>=1]
]
```

We use the Unequal clause to good effect. We don’t want the leading letters to be zero, so we add inequalities for that.

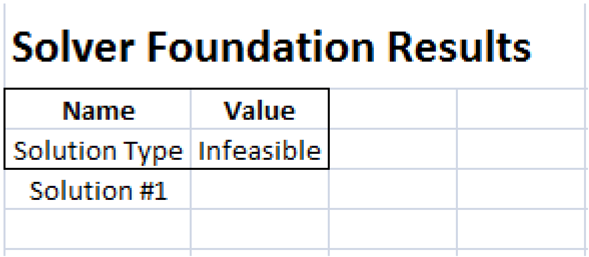
In (Knuth, 2004) a very large alphametic is shown:

```
          AN
ACCELERATING
 INFERENTIAL
 ENGINEERING
        TALE
       ELITE
       GRANT
         FEE
          ET
      CETERA
------------
  ARTIFICIAL
INTELLIGENCE
```

This can be coded along the same lines:

```
Model[
  Decisions[Integers[0,9],A,N,C,E,L,R,T,I,G,F],
  Constraints[
    10*A+N+
    1e11*A+1e10*C+1e9*C+1e8*E+1e7*L+1e6*E+1e5*R+1e4*A+1e3*T+100*I+10*N+G+
    1e10*I+1e9*N+1e8*F+1e7*E+1e6*R+1e5*E+1e4*N+1e3*T+100*I+10*A+L+
    1e10*E+1e9*N+1e8*G+1e7*I+1e6*N+1e5*E+1e4*E+1e3*R+100*I+10*N+G+
    1e3*T+100*A+10*L+E+
    1e4*E+1e3*L+100*I+10*T+E+
    1e4*G+1e3*R+100*A+10*N+T+
    100*F+10*E+E+
    10*E+T+
    1e5*C+1e4*E+1e3*T+100*E+10*R+A
       ==
    1e9*A+1e8*R+1e7*T+1e6*I+1e5*F+1e4*I+1e3*C+100*I+10*A+L+
    1e11*I+1e10*N+1e9*T+1e8*E+1e7*L+1e6*L+1e5*I+1e4*G+1e3*E+100*N+10*C+E,
    A>=1, I>=1,E>=1,T>=1,G>=1,F>=1,C>=1,
    Unequal[A,N,C,E,L,R,T,I,G,F]
  ]
]
```

The CSP solver uses some rational arithmetic so there is hope we can solve this as stated. Unfortunately we get:



Solver Foundation Results

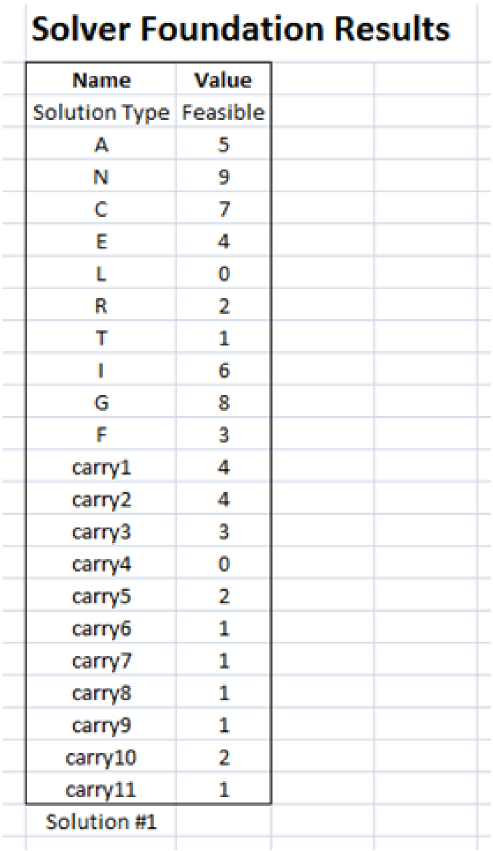
Name	Value
Solution Type	Infeasible
Solution #1	

The reason is that intermediate results overflow the integer range. The CSP solver will only search where all terms are within a valid integer range.

There is an alternative formulation where we add up digits slice by slice just as taught in elementary school. We use an explicit carry mechanism in the addition.

```
Model[
  Decisions[Integers[0,9],A,N,C,E,L,R,T,I,G,F],
  Decisions[Integers[0,Infinity],carry1,carry2,carry3,carry4,carry5,carry6,
         carry7,carry8,carry9,carry10,carry11],
  Constraints[
    N + G + L + G + E + E + T + E + T + A == L + E + 10*carry1,
    A + N + A + N + L + T + N + E + E + R + carry1 == A + C + 10*carry2,
    I + I + I + A + I + A + F + E + carry2 == I + N + 10*carry3,
    T + T + R + T + L + R + T + carry3 == C + E + 10*carry4,
    A + N + E + E + G + E + carry4 == I + G + 10*carry5,
    R + E + E + C + carry5 == F + I + 10*carry6,
    E + R + N + carry6 == I + L + 10*carry7,
    L + E + I + carry7 == T + L + 10*carry8,
    E + F + G + carry8 == R + E + 10*carry9,
    C + N + N + carry9 == A + T + 10*carry10,
    C + I + E + carry10 == N + 10*carry11,
    A + carry11 == I,
    A>=1, I>=1,E>=1,T>=1,G>=1,F>=1,C>=1,
    Unequal[A,N,C,E,L,R,T,I,G,F]
  ]
]
```

This model solves the problem quickly, giving the solution:



Solver Foundation Results

Name	Value
Solution Type	Feasible
A	5
N	9
C	7
E	4
L	0
R	2
T	1
I	6
G	8
F	3
carry1	4
carry2	4
carry3	3
carry4	0
carry5	2
carry6	1
carry7	1
carry8	1
carry9	1
carry10	2
carry11	1

Solution #1

This example shows the importance of trying different formulations. This is easier with a modeling language than with coding in a programming language.

5.11 DATA ENVELOPMENT ANALYSIS

Data envelopment analysis or DEA is an LP based technique for evaluating the relative efficiency of Decision Making Units (DMU's). In many cases the performance of non-profit and government organizational units is very difficult to compare: their outputs are not readily comparable and no monetary value can be easily assigned to inputs or outputs. With this technique, one can make draw some conclusions, using concept related to an efficient frontier known from quadratic programming applications in finance. It is a non-parametric method: we don't need an explicit specification of the functional relationship between inputs and outputs (i.e. a production function).

We assume that each DMU *j* has multiple inputs $x_{i,j}$ and multiple outputs $y_{k,j}$. A relative efficiency measure is defined by:

$$\text{Efficiency} = \frac{\sum_k u_k y_{k,j}}{\sum_i v_i x_{i,j}}$$

where *u* and *v* are weights. Often the efficiency is scaled so that it ranges from [0, 1].

The weights form a problem: setting a uniform value for them over all DMU's is rather arbitrary. The main idea behind DEA, is that we allow each DMU *j0* to set its own weights. It can use the following optimization problem for that: maximize the efficiency of DMU *j0* subject to the condition that all efficiencies of other DMU's remain less than or equal to 1. I.e.

$$\begin{aligned} \text{maximize}_{u,v}\ \ \vartheta_0 &= \frac{\sum_k u_k y_{k,j0}}{\sum_i v_i x_{i,j0}} \\ \text{subject to}\ \ \frac{\sum_k u_k y_{k,j}}{\sum_i v_i x_{i,j}} &\le 1\ \forall j \\ u_k, v_i &\ge 0 \end{aligned}$$

This is not an LP however. A simple work around is to fix the denominator to a constant value, e.g. 1.0, which can be interpreted as setting a constraint on the weights v_i (often weights are normalized to add up to one; this can be considered as a slightly more complex normalization). This results in:

$$\begin{gathered} \text{maximize}_{u,v}\ \ \sum\nolimits_k u_k y_{k,j0} \\ \text{subject to}\ \ \sum\nolimits_i v_i x_{i,j0} = 1 \\ \sum\nolimits_k u_k y_{k,j} \le \sum\nolimits_i v_i x_{i,j}\quad \forall j \\ u_k, v_i \ge 0 \end{gathered}$$

It is noted that *x* and *y* are no decision variables but rather data. The decision variables are the weights *u* and *v*.

Note that there are many variants formulated for this problem.

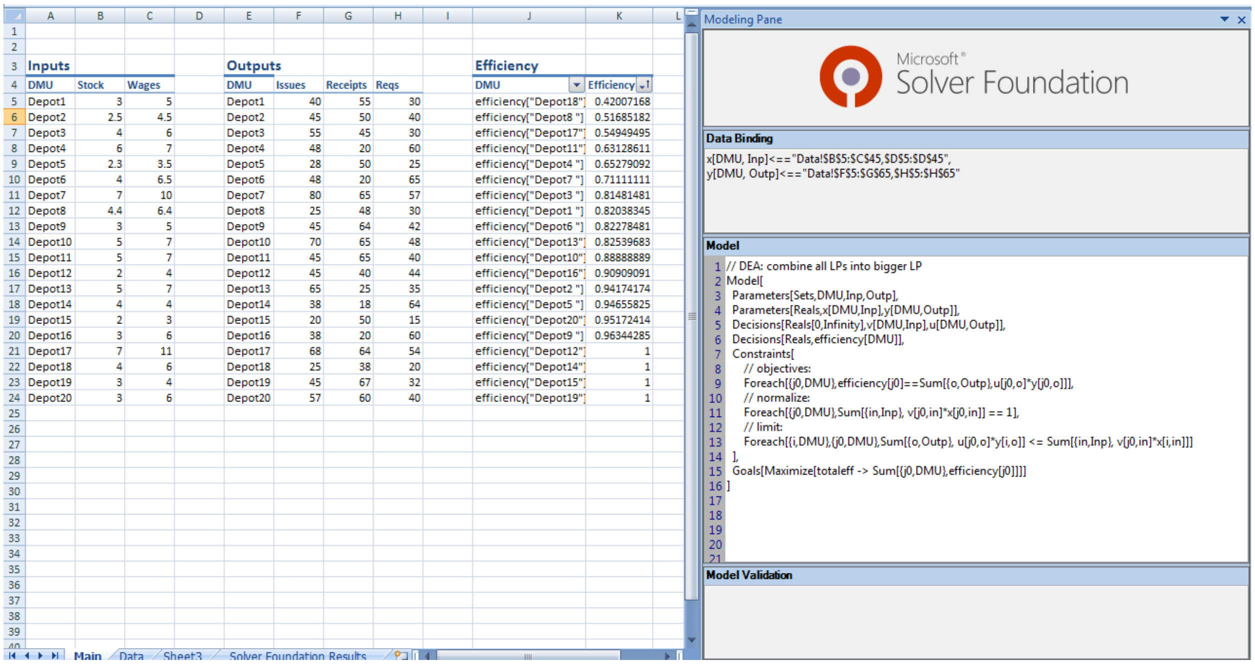
When solving this problem for a data set, we form a little LP for every DMU *j0*. When using the Excel plug-in it is not directly possible to use different models. So instead of solving a series of small LP's we solve one larger problem. This can be depicted as:

$$\begin{array}{llllll} \text{min} & c'_1 x_1 & +c'_2 x_2 & \cdots & +c'_K x_K \\ \text{s.t.} & A_1 x_1 = b_1 & & & \\ & & A_2 x_2 = b_2 & & \\ & & & \ddots & \\ & & & & A_K x_K = b_K \end{array}$$

The OML model can look like:

```
// DEA: combine all LPs into bigger LP
Model[
  Parameters[Sets,DMU,Inp,Outp],
  Parameters[Reals,x[DMU,Inp],y[DMU,Outp]],
  Decisions[Reals[0,Infinity],v[DMU,Inp],u[DMU,Outp]],
  Decisions[Reals,efficiency[DMU]],
  Constraints[
      // objectives:
      Foreach[{j0,DMU},efficiency[j0]==Sum[{o,Outp},u[j0,o]*y[j0,o]]],
      // normalize:
      Foreach[{j0,DMU},Sum[{in,Inp}, v[j0,in]*x[j0,in]] == 1],
      // limit:
      Foreach[{i,DMU},{j0,DMU},Sum[{o,Outp}, u[j0,o]*y[i,o]] <= Sum[{in,Inp}, v[j0,in]*x[i,in]]]
  ],
  Goals[Maximize[totaleff -> Sum[{j0,DMU},efficiency[j0]]]]
]
```

A complete Excel sheet can look like:



Inputs				Outputs					Efficiency	
DMU	Stock	Wages		DMU	Issues	Receipts	Reqs		DMU	Efficiency
Depot1	3	5		Depot1	40	55	30		efficiency["Depot18"]	0.42007168
Depot2	2.5	4.5		Depot2	45	50	40		efficiency["Depot8 "]	0.51685182
Depot3	4	6		Depot3	55	45	30		efficiency["Depot17"]	0.54949495
Depot4	6	7		Depot4	48	20	60		efficiency["Depot11"]	0.63128611
Depot5	2.3	3.5		Depot5	28	50	25		efficiency["Depot4 "]	0.65279092
Depot6	4	6.5		Depot6	48	20	65		efficiency["Depot7 "]	0.71111111
Depot7	7	10		Depot7	80	65	57		efficiency["Depot3 "]	0.81481481
Depot8	4.4	6.4		Depot8	25	48	30		efficiency["Depot1 "]	0.82038345
Depot9	3	5		Depot9	45	64	42		efficiency["Depot6 "]	0.82278481
Depot10	5	7		Depot10	70	65	48		efficiency["Depot13"]	0.82539683
Depot11	5	7		Depot11	45	65	40		efficiency["Depot10"]	0.88888889
Depot12	2	4		Depot12	45	40	44		efficiency["Depot16"]	0.90909091
Depot13	5	7		Depot13	65	25	35		efficiency["Depot2 "]	0.94174174
Depot14	4	4		Depot14	38	18	64		efficiency["Depot5 "]	0.94655825
Depot15	2	3		Depot15	20	50	15		efficiency["Depot20"]	0.95172414
Depot16	3	6		Depot16	38	20	60		efficiency["Depot9 "]	0.96344285
Depot17	7	11		Depot17	68	64	54		efficiency["Depot12"]	1
Depot18	4	6		Depot18	25	38	20		efficiency["Depot14"]	1
Depot19	3	4		Depot19	45	67	32		efficiency["Depot15"]	1
Depot20	3	6		Depot20	57	60	40		efficiency["Depot19"]	1

Data Binding

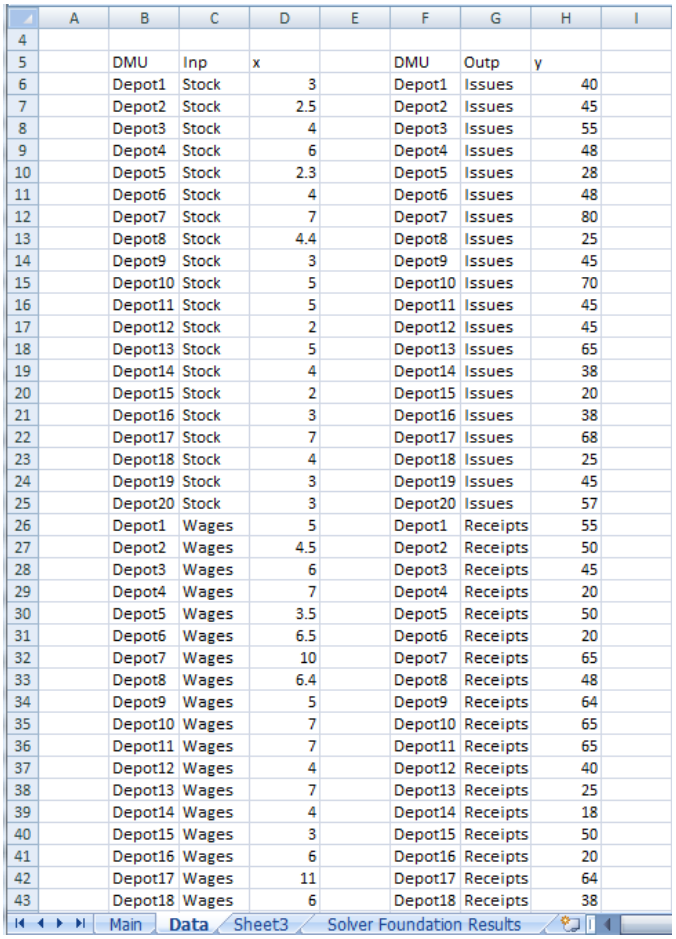
```
x[DMU, Inp]<=="Data!$B$5:$C$45,$D$5:$D$45",
y[DMU, Outp]<=="Data!$F$5:$G$65,$H$5:$H$65"
```

Model

```
// DEA: combine all LPs into bigger LP
Model[
 Parameters[Sets,DMU,Inp,Outp],
 Parameters[Reals,x[DMU,Inp],y[DMU,Outp]],
 Decisions[Reals[0,Infinity],v[DMU,Inp],u[DMU,Outp]],
 Decisions[Reals,efficiency[DMU]],
 Constraints[
    // objectives:
    Foreach[{j0,DMU},efficiency[j0]==Sum[{o,Outp},u[j0,o]*y[j0,o]]],
    // normalize:
    Foreach[{j0,DMU},Sum[{in,Inp}, v[j0,in]*x[j0,in]] == 1],
    // limit:
    Foreach[{i,DMU},{j0,DMU},Sum[{o,Outp}, u[j0,o]*y[i,o]] <= Sum[{in,Inp}, v[j0,in]*x[i,in]]]
 ],
 Goals[Maximize[totaleff -> Sum[{j0,DMU},efficiency[j0]]]]
]
```

Model Validation

To make it easy for MSF to read the data it is organized as:



	DMU	Inp	x		DMU	Outp	y
6	Depot1	Stock	3		Depot1	Issues	40
7	Depot2	Stock	2.5		Depot2	Issues	45
8	Depot3	Stock	4		Depot3	Issues	55
9	Depot4	Stock	6		Depot4	Issues	48
10	Depot5	Stock	2.3		Depot5	Issues	28
11	Depot6	Stock	4		Depot6	Issues	48
12	Depot7	Stock	7		Depot7	Issues	80
13	Depot8	Stock	4.4		Depot8	Issues	25
14	Depot9	Stock	3		Depot9	Issues	45
15	Depot10	Stock	5		Depot10	Issues	70
16	Depot11	Stock	5		Depot11	Issues	45
17	Depot12	Stock	2		Depot12	Issues	45
18	Depot13	Stock	5		Depot13	Issues	65
19	Depot14	Stock	4		Depot14	Issues	38
20	Depot15	Stock	2		Depot15	Issues	20
21	Depot16	Stock	3		Depot16	Issues	38
22	Depot17	Stock	7		Depot17	Issues	68
23	Depot18	Stock	4		Depot18	Issues	25
24	Depot19	Stock	3		Depot19	Issues	45
25	Depot20	Stock	3		Depot20	Issues	57
26	Depot1	Wages	5		Depot1	Receipts	55
27	Depot2	Wages	4.5		Depot2	Receipts	50
28	Depot3	Wages	6		Depot3	Receipts	45
29	Depot4	Wages	7		Depot4	Receipts	20
30	Depot5	Wages	3.5		Depot5	Receipts	50
31	Depot6	Wages	6.5		Depot6	Receipts	20
32	Depot7	Wages	10		Depot7	Receipts	65
33	Depot8	Wages	6.4		Depot8	Receipts	48
34	Depot9	Wages	5		Depot9	Receipts	64
35	Depot10	Wages	7		Depot10	Receipts	65
36	Depot11	Wages	7		Depot11	Receipts	65
37	Depot12	Wages	4		Depot12	Receipts	40
38	Depot13	Wages	7		Depot13	Receipts	25
39	Depot14	Wages	4		Depot14	Receipts	18
40	Depot15	Wages	3		Depot15	Receipts	50
41	Depot16	Wages	6		Depot16	Receipts	20
42	Depot17	Wages	11		Depot17	Receipts	64
43	Depot18	Wages	6		Depot18	Receipts	38

5.12 QUADRATIC PROGRAMMING AND PORTFOLIO MODELS

In this section we discuss a few formulations for the portfolio optimization problem.

5.12.1 MEAN-VARIANCE PORTFOLIO SELECTION AND EFFICIENT FRONTIERS

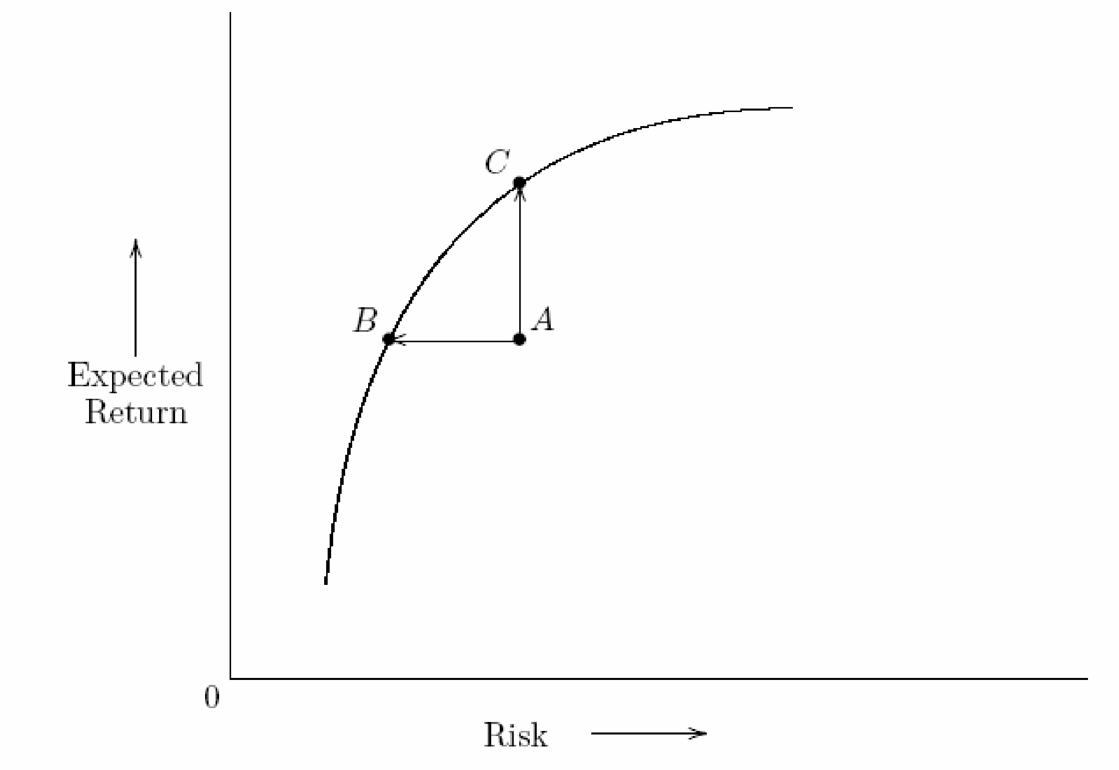
A standard mean-value portfolio model can be formulated as:

$$\begin{gathered}\min x'Qx \\ r'x \geq \rho \\ \sum x_i = 1 \\ x \geq 0\end{gathered}$$

i.e. we minimize risk, subject a minimum return and a budget constraint. This problem can also be changed into a multi-objective problem, where we use a weighting function to combine both objectives: minimize risk and maximize return:

$$\begin{gathered}\min x'Qx - \lambda r'x \\ \sum x_i = 1 \\ x \geq 0\end{gathered}$$

When we plot the optimal values for the variance and the return for different values of λ we get something like:



The points on the curve are all “efficient” portfolios. One could devise a portfolio below the curve (e.g. portfolio A). However this is not an efficient portfolio: there is another portfolio B with the same return but less risk and there is another portfolio C with the same risk but better expected return.

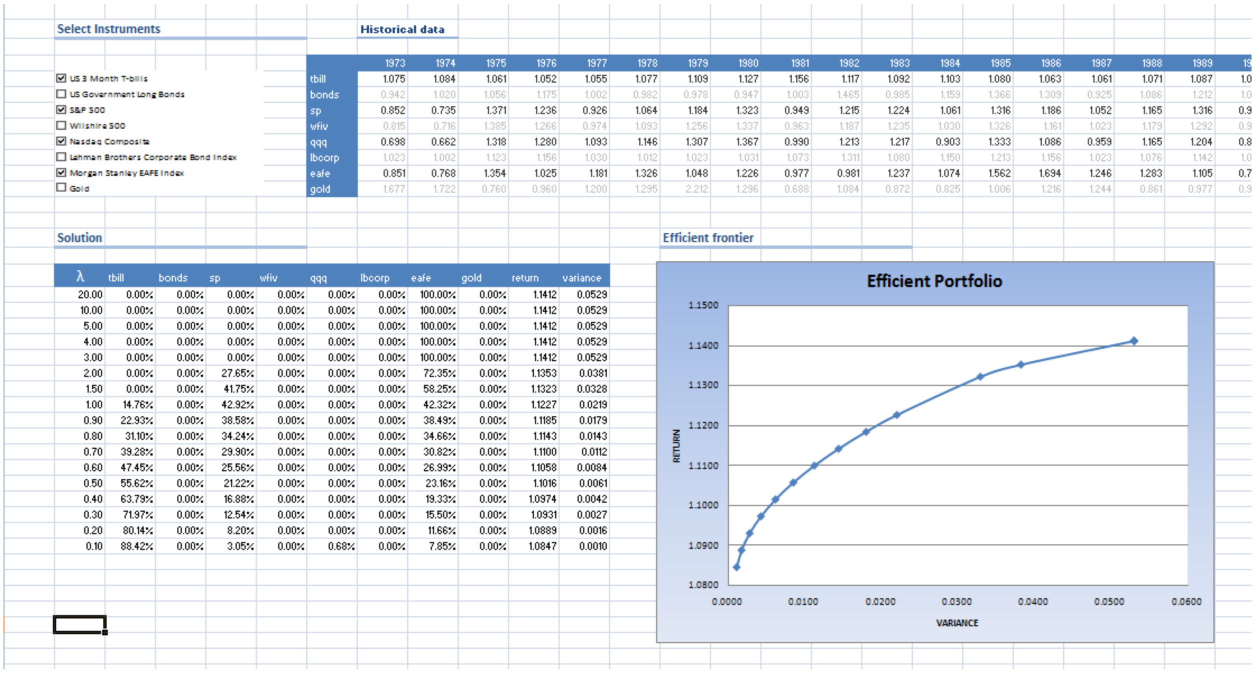
The simplest way to calculate this is solving a series of QP’s (a better way would be to use some parametric technique). This is not really possible in OML, but we can combine all QP’s together and form a large QP. The structure would be:

$$\begin{array}{ll} \min & x'_1Qx_1 - \lambda_1\mu'x_1 \quad +x'_2Qx_2 - \lambda_2\mu'x_2 \quad \cdots \quad +x'_KQx_K - \lambda_K\mu'x_K \\ \text{s.t.} & \sum_i x_{1,i} = 1 \\ & \qquad\qquad \sum_i x_{2,i} = 1 \\ & \qquad\qquad\qquad\qquad \ddots \\ & \qquad\qquad\qquad\qquad\qquad \sum_i x_{K,i} = 1 \end{array}$$

We omitted here the x≥0 condition. The OML model can look like:

```
Model[
   Parameters[Sets,I,K],
   Parameters[Integers,Selected[I]],
   Parameters[Reals,Return[I]],
   Parameters[Reals,Lambda[K]],
   Parameters[Reals,Covar[I,I]],

   Decisions[Reals[0,Infinity],Alloc[K,I]],
   Constraints[
       Foreach[{k,K},FilteredSum[{i,I},Selected[i]>0,Alloc[k,i]]==1]
   ],

   Goals[Minimize["Overall"->
         Sum[{k,K},
              FilteredSum[{i,I},Selected[i]>0,
                   FilteredSum[{j,I},Selected[j]>0,Alloc[k,i]*Covar[i,j]*Alloc[k,j]]]
              - Lambda[k]*FilteredSum[{i,I},Selected[i]>0,Return[i]*Alloc[k,i]]
           ]
   ]]
]
```

Here `Selected` is a parameter indicating whether an instrument can be part of the portfolio. The Excel GUI can look like:



Select Instruments

☑ US 3 Month T-bills
□ US Government Long Bonds
☑ S&P 500
□ Wilshire 500
☑ Nasdaq Composite
□ Lehman Brothers Corporate Bond Index
☑ Morgan Stanley EAFE Index
□ Gold

Historical data

	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	19
tbill	1.075	1.084	1.061	1.052	1.055	1.077	1.109	1.127	1.156	1.117	1.092	1.103	1.080	1.063	1.061	1.071	1.087	1.0
bonds	0.942	1.020	1.056	1.175	1.002	0.982	0.978	0.947	1.003	1.465	0.985	1.159	1.366	1.309	0.925	1.086	1.212	1.0
sp	0.852	0.735	1.371	1.236	0.926	1.064	1.184	1.323	0.949	1.215	1.224	1.061	1.316	1.186	1.052	1.165	1.316	0.9
wfiv	0.815	0.716	1.385	1.266	0.974	1.093	1.256	1.337	0.963	1.187	1.235	1.030	1.326	1.161	1.023	1.179	1.292	0.9
qqq	0.698	0.662	1.318	1.280	1.093	1.146	1.307	1.367	0.990	1.213	1.217	0.903	1.333	1.086	0.959	1.165	1.204	0.8
lbcorp	1.023	1.002	1.123	1.156	1.030	1.012	1.023	1.031	1.073	1.311	1.080	1.150	1.213	1.156	1.023	1.076	1.142	1.0
eafe	0.851	0.768	1.354	1.025	1.181	1.326	1.048	1.226	0.977	0.981	1.237	1.074	1.562	1.694	1.246	1.283	1.105	0.7
gold	1.677	1.722	0.760	0.960	1.200	1.295	2.212	1.296	0.688	1.084	0.872	0.825	1.006	1.216	1.244	0.861	0.977	0.9

Solution

λ	tbill	bonds	sp	wfiv	qqq	lbcorp	eafe	gold	return	variance
20.00	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	1.1412	0.0529
10.00	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	1.1412	0.0529
5.00	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	1.1412	0.0529
4.00	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	1.1412	0.0529
3.00	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	1.1412	0.0529
2.00	0.00%	0.00%	27.65%	0.00%	0.00%	0.00%	72.35%	0.00%	1.1353	0.0381
1.50	0.00%	0.00%	41.75%	0.00%	0.00%	0.00%	58.25%	0.00%	1.1323	0.0328
1.00	14.76%	0.00%	42.92%	0.00%	0.00%	0.00%	42.32%	0.00%	1.1227	0.0219
0.90	22.93%	0.00%	38.58%	0.00%	0.00%	0.00%	38.49%	0.00%	1.1185	0.0179
0.80	31.10%	0.00%	34.24%	0.00%	0.00%	0.00%	34.66%	0.00%	1.1143	0.0143
0.70	39.28%	0.00%	29.90%	0.00%	0.00%	0.00%	30.82%	0.00%	1.1100	0.0112
0.60	47.45%	0.00%	25.56%	0.00%	0.00%	0.00%	26.99%	0.00%	1.1058	0.0084
0.50	55.62%	0.00%	21.22%	0.00%	0.00%	0.00%	23.16%	0.00%	1.1016	0.0061
0.40	63.79%	0.00%	16.88%	0.00%	0.00%	0.00%	19.33%	0.00%	1.0974	0.0042
0.30	71.97%	0.00%	12.54%	0.00%	0.00%	0.00%	15.50%	0.00%	1.0931	0.0027
0.20	80.14%	0.00%	8.20%	0.00%	0.00%	0.00%	11.66%	0.00%	1.0889	0.0016
0.10	88.42%	0.00%	3.05%	0.00%	0.68%	0.00%	7.85%	0.00%	1.0847	0.0010

Efficient frontier

If we denote the data by $d_{i,t}$ then we can define

$$\mu_i = \frac{1}{T}\sum_{t=1}^{T} d_{i,t}$$

Then the mean-adjusted returns are:

$$d'_{i,t} = d_{i,t} - \mu_i$$

The covariances are defined by:

$$q_{i,j} = \frac{1}{T}\sum_{t=1}^{T} d'_{i,t} d'_{j,t}$$

We now can rewrite:

$$x'Qx = \sum_{i,j} x_i q_{i,j} x_j = \frac{1}{T}\sum_{t}\left[\left(\sum_i x_i d'_{i,t}\right)\left(\sum_j x_j d'_{j,t}\right)\right] = \frac{1}{T}\sum_t w_t^2$$

where

$$w_t = \sum_k x_k d'_{k,t}$$

So the QP becomes:

$$
\begin{gathered}
\min \frac{1}{T}\sum w_t^2 - \lambda\, r'x \\
\sum x_i \,= 1 \\
w_t \,= \sum_k x_k\, d'_{k,t} \\
x \geq 0
\end{gathered}
$$

This formulation is often beneficial if there are more instruments than time periods. This is not unusual as older historical data are assumed to have little predictive value. Also note that this formulation suggests an easy linearization:

$$\sum w_t^2 \approx \sum |w_t|$$

The quadratic reformulation can be coded in OML as:

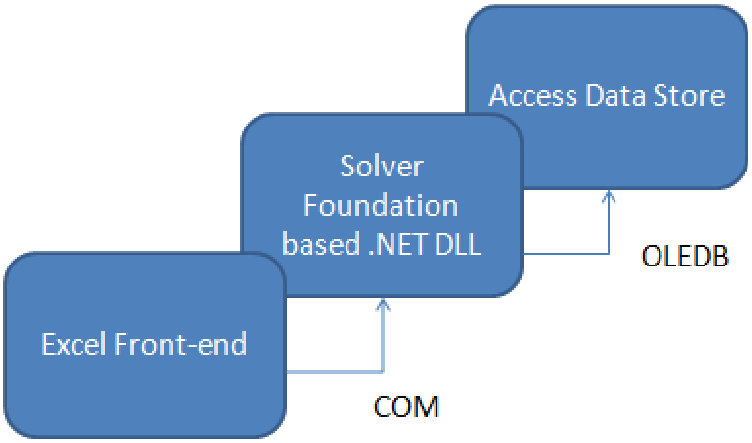
```
Model[
   Parameters[Sets,I,K,T],
   Parameters[Integers,Selected[I]],
   Parameters[Reals,Return[I]],
   Parameters[Reals,Lambda[K]],
   Parameters[Reals,MeanAdjRet[I,T]],
   Parameters[Integers,TT[]],

   Decisions[Reals[0,Infinity],Alloc[K,I]],
   Decisions[Reals,w[K,T]],

   Constraints[
       Foreach[{k,K},FilteredSum[{i,I},Selected[i]>0,Alloc[k,i]]==1],
       Foreach[{k,K},{t,T},w[k,t]==FilteredSum[{i,I},Selected[i]>0,MeanAdjRet[i,t]*Alloc[k,i]]]
   ],

   Goals[Minimize["Overall"->
         Sum[{k,K},
              Sum[{t,T},w[k,t]^2]/TT[]
              - Lambda[k]*FilteredSum[{i,I},Selected[i]>0,Return[i]*Alloc[k,i]]
           ]
   ]]
]
```

5.12.2 LP APPROXIMATIONS AND .NET DLL'S

The Excel plug-in allows only for one single model to be solved. In the previous section we showed how to solve a number of QP models as one big QP model, so we could use the Excel plug-in. In this section we demonstrate how to solve different models from Excel using some VBA code and a .NET DLL that handles the interface to Microsoft Solver Foundation.

The architecture of the application is as follows:

Architecture



The Excel front-end is used to host some buttons to run the model and to present the results. The DLL is written in C# and holds the QP models and does the calls to MSF. Finally the database is a simple MS Access database with a few queries that implement the data manipulation steps needed to prepare the data for the optimization models.

The model we are going to solve is:

$$\begin{gathered}\min \frac{\lambda}{T} \sum w_t^2 - r'x \\ \sum x_i = 1 \\ w_t = \sum_k x_k d'_{k,t} \\ x \geq 0\end{gathered}$$

where

$$d'_{i,t} = d_{i,t} - \mu_i$$

In addition we can solve the linear approximation:

$$\begin{gathered}\min \frac{\lambda}{T} \sum |w_t| - r'x \\ \sum x_i = 1 \\ w_t = \sum_k x_k d'_{k,t} \\ x \geq 0\end{gathered}$$

The absolute value can be linearized using a standard variable splitting technique:

$$\min \frac{\lambda}{T}\sum(y_t^+ + y_t^-) - r'x$$

$$\sum x_i = 1$$

$$w_t = \sum_k x_k d'_{k,t}$$

$$y_t^+ - y_t^- = w_t$$

$$x, y^+, y^- \geq 0$$

The mean-adjusted returns are calculated in the database as follows:

Query:mean	Query:adjreturns
SELECT instrument, avg(return) AS mean	SELECT returns.instrument, returns.year, (return-mean) AS adjreturn
FROM returns	FROM returns, mean
GROUP BY instrument;	WHERE (((returns.instrument)=[mean].[instrument]));

The code looks as follows:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using Microsoft.SolverFoundation.Services;
using System.Data;
using System.Data.OleDb;
using System.Data.Linq;


namespace OMLFromCSharp
{
    /// <summary>
    /// Solve portfolio models
    /// </summary>
    [ClassInterface(ClassInterfaceType.AutoDual)]
    public class Portfolio
    {
        /// <summary>
        /// Version of the model
        /// </summary>
        string versionString = @"PORTFOLIO MODEL 1.0";

        /// <summary>
        /// Holds the QP model
        /// </summary>
        string strModelQP = @"Model[
              Parameters[Sets,I,T],
              Parameters[Reals,AdjRet[T,I],Mean[I]],
              Parameters[Reals,Lambda={0}],
              Parameters[Integers,NumT={1}],

              Decisions[Reals[0,Infinity],x[I]],
              Decisions[Reals,w[T],Return],

              Constraints[
                  Sum[{{i,I}},x[i]]==1,
                  Foreach[{{t,T}},w[t]==Sum[{{i,I}},AdjRet[t,i]*x[i]]],
                  Return==Sum[{{i,I}},Mean[i]*x[i]]
              ],

              Goals[Minimize[Obj->Lambda*Sum[{{t,T}},w[t]^2]/NumT-Return]]
           ]";
        /// <summary>
        /// Holds the LP model
        /// </summary>
        string strModelLP = @"Model[
              Parameters[Sets,I,T],
              Parameters[Reals,AdjRet[T,I],Mean[I]],
              Parameters[Reals,Lambda={0}],
              Parameters[Integers,NumT={1}],

              Decisions[Reals[0,Infinity],x[I],yplus[T],ymin[T]],
              Decisions[Reals,w[T],Return],

              Constraints[
                  Sum[{{i,I}},x[i]]==1,
                  Foreach[{{t,T}},w[t]==Sum[{{i,I}},AdjRet[t,i]*x[i]]],
                  Return==Sum[{{i,I}},Mean[i]*x[i]],
                  Foreach[{{t,T}},yplus[t]-ymin[t]==w[t]]
              ],

              Goals[Minimize[Obj->Lambda*Sum[{{t,T}},yplus[t]+ymin[t]]/NumT-Return]]
           ]";

        /// <summary>
        ///  SFS
        /// </summary>
        SolverContext context;

        /// <summary>
        /// msf solution
        /// </summary>
        Solution solution;

        /// <summary>
        /// Number of instruments we can invest in
        /// </summary>
        int numInstruments;

        /// <summary>
        /// number of time periods
        /// </summary>
        int numYears;

        /// <summary>
        /// lambda value used
        /// </summary>
        double lambda;

        /// <summary>
        /// Connection string for MS Access
        /// Use x86 architecture!
        /// </summary>
        string connection = @"Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=C:\projects\ms\OMLFromCSharp\OMLFromCSharp\bin\Debug\portfolio1.accdb;Persist Security
Info=False;";

        /// <summary>
        /// Connection object
        /// </summary>
        OleDbConnection conn;

        /// <summary>
        /// list of instruments
        /// </summary>
        string[] instruments;

        /// <summary>
        ///  Constructor
        /// </summary>
        public Portfolio()
        {
            context = SolverContext.GetContext();
        }

        /// <summary>
        /// get query result as DataSet
        /// </summary>
        /// <param name="query">query as string</param>
        /// <returns></returns>
        private DataSet SelectOleDbSrvRows(string query)
        {
            DataSet ds = new DataSet();
            OleDbDataAdapter adapter = new OleDbDataAdapter();
            adapter.SelectCommand = new OleDbCommand(query, conn);
            adapter.Fill(ds);
            return ds;
        }

        /// <summary>
        /// Perform some magic to make sure the query output arrives in OML model.
        /// </summary>
        /// <param name="p">OML/SFS parameter</param>
        /// <param name="query">database query</param>
        /// <param name="valueColumn">column with values</param>
        /// <param name="IndexColumns">columns with indices</param>
        private void setBinding(Parameter p, string query, string valueColumn, params string[]
IndexColumns)
        {
            DataSet ds = SelectOleDbSrvRows(query);
            DataTable dt = ds.Tables[0];
            p.SetBinding(dt.AsEnumerable(), valueColumn, IndexColumns);
        }


        /// <summary>
        /// Load parameters into OML model
        /// </summary>
        private void loadParams()
        {
            foreach (Parameter p in context.CurrentModel.Parameters)
            {
                switch (p.Name)
                {
                    case "AdjRet":
                        setBinding(p, "select [year],instrument,adjreturn from adjreturns",
                            "adjreturn", "year", "instrument");
                        break;
                    case "Mean":
                        setBinding(p, "select instrument,mean from mean",
                            "mean", "instrument");
                        break;
                }

            }
        }

        /// <summary>
        /// get some counts from the database
        /// </summary>
        private void loadScalars()
        {
            OleDbCommand dbcommand = new OleDbCommand("select [value] from counts where
[key]='years'", conn);
            numYears = (int)dbcommand.ExecuteScalar();
            dbcommand = new OleDbCommand("select [value] from counts where [key]='instruments'",
conn);
            numInstruments = (int)dbcommand.ExecuteScalar();
        }

        /// <summary>
        /// get number of time periods
        /// </summary>
        /// <returns></returns>
        public int getNumt()
        {
            return numYears;
        }

        /// <summary>
        /// get number of instruments
        /// </summary>
        /// <returns></returns>
        public int getNumInstruments()
        {
            return numInstruments;
        }

        /// <summary>
        /// get instruments as
        /// </summary>
        /// <returns></returns>
        private string[] getInstrumentsFromDB()
        {
            List<string> s = new List<string>();

            OleDbCommand dbcommand = new OleDbCommand("select instrument from instruments",
conn);
            OleDbDataReader reader = dbcommand.ExecuteReader();
            while (reader.Read())
            {
                s.Add(reader[0].ToString());
            }
            reader.Close();
            return s.ToArray();
        }

        /// <summary>
        /// get some data from DB
        /// </summary>
        private void initDataFromDB()
        {
            loadScalars();
            instruments = getInstrumentsFromDB();
        }

        /// <summary>
        /// Solve the QP problem
        /// </summary>
        /// <param name="plambda">Value of lambda</param>
        /// <returns>true on success</returns>
        public Boolean SolveQP(double plambda)
        {
            try
            {
                lambda = plambda;
                context.ClearModel();
                if (conn == null)
                {
                    conn = new OleDbConnection(connection);
                    conn.Open();
                    initDataFromDB();
                }
                string s = String.Format(strModelQP, lambda, numYears);
                context.LoadModel(FileFormat.OML, new StringReader(s));
                loadParams();
                solution = context.Solve();
                return true;
            }
            catch(Exception e)
            {
                MessageBox.Show(string.Format("Exception: {0}", e.Message));
                return false;
            }
        }

        /// <summary>
        /// Solve the QP problem
        /// </summary>
        /// <param name="plambda">Value of lambda</param>
        /// <returns>true on success</returns>
        public Boolean SolveLP(double plambda)
        {
            try
            {
                lambda = plambda;
                context.ClearModel();
                if (conn == null)
                {
                    conn = new OleDbConnection(connection);
                    conn.Open();
                    initDataFromDB();
                }
                string s = String.Format(strModelLP, lambda, numYears);
                context.LoadModel(FileFormat.OML, new StringReader(s));
                loadParams();
                solution = context.Solve();
                return true;
            }
            catch (Exception e)
            {
                MessageBox.Show(string.Format("Exception: {0}", e.Message));
                return false;
            }
        }

        /// <summary>
        /// get solution vector
        /// </summary>
        /// <param name="varx">x vector</param>
        /// <param name="returnx">portfolio returns</param>
        /// <param name="riskx">risk term</param>
        public void getSolution(ref double[] varx, ref double returnx, ref double riskx)
        {
            try
            {
                context.PropagateDecisions();
                int n = instruments.Count();
                varx = new double[n];
                Decision x = context.CurrentModel.Decisions.First(d => d.Name == "x");
                for (int i=0; i < n; ++i)
                    varx[i] = x.GetDouble(instruments[i]);
                Decision xreturn = context.CurrentModel.Decisions.First(d => d.Name == "Return");
                returnx = xreturn.ToDouble();
                double obj = solution.Goals.ElementAt(0).ToDouble();
                riskx = (obj + returnx)*numYears/lambda;

            }
            catch (Exception e)
            {
                MessageBox.Show(string.Format("Exception: {0}", e.Message));
            }

        }

        /// <summary>
        /// get name
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        public String xname(int i)
        {
            return instruments[i];
        }
        /// <summary>
        /// Returns the model version
        /// </summary>
        /// <returns>Version string</returns>
        public String version()
        {
            return versionString;
        }


    }
}
```

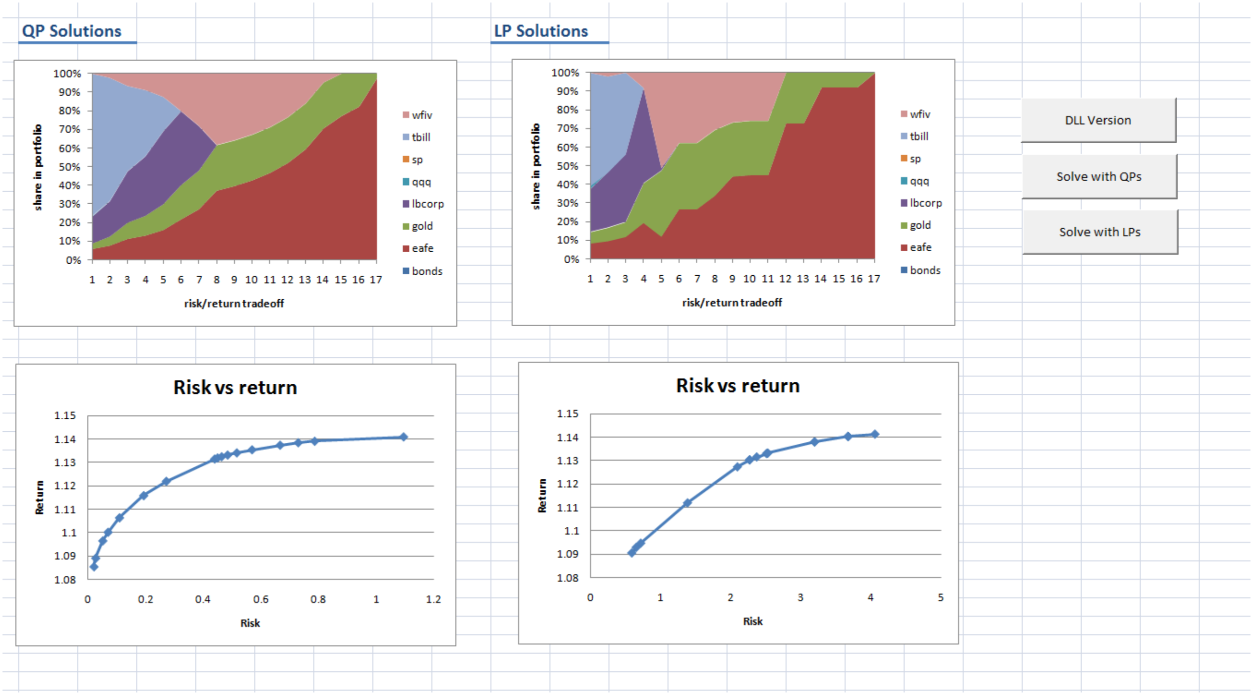
The model follows largely the example discussed in section 4.1. The line

```
[ClassInterface(ClassInterfaceType.AutoDual)]
```

will instruct the compiler to generate a COM interface so we can call the public members from some VBA code that resides in an Excel spreadsheet. The DLL needs to be registered as a COM component (Visual Studio can do this automatically for you). The spreadsheet interface looks like:

```
Dim oml As New OMLFromCSharp.Portfolio

Sub demoSolve()
   Dim L As Variant
   L = Range("lambdas").Value
   Dim i As Integer
   Dim ok As Boolean

   For i = LBound(L, 1) To UBound(L, 1)
       ok = oml.SolveQP(L(i, 1))
   Next i
End Sub

Public Sub showVersion()
   MsgBox oml.Version
End Sub

Public Sub solveQPs()
   Dim L As Variant
   L = Range("lambdas").Value
   Dim i As Integer
   Dim j As Integer
   Dim n As Long
   n = 0
   Dim x() As Double
   Dim retx As Double
   Dim riskx As Double
   Dim ok As Boolean


   For i = LBound(L, 1) To UBound(L, 1)
         ok = oml.SolveQP(L(i, 1))
         If (Not ok) Then
           Exit Sub
         End If
         If (n = 0) Then
            n = oml.getNumInstruments()
            ReDim x(n)
         End If
         Call oml.getSolution(x, retx, riskx)
         Range("Solution!C10").Cells(i, 1) = retx
         Range("Solution!C10").Cells(i, 2) = riskx
         For j = 0 To n - 1
            Range("Solution!C10").Cells(i, j + 3) = x(j)
         Next j

   Next i

   Dim xname() As String
   ReDim xname(n)
   For i = 0 To n - 1
      xname(i) = oml.xname(i)
      Range("Solution!C10").Cells(0, i + 3) = xname(i)
   Next i


End Sub

Public Sub solveLPs()
   Dim L As Variant
   L = Range("LPlambdas").Value
   Dim i As Integer
   Dim j As Integer
   Dim n As Long
   n = 0
   Dim x() As Double
   Dim retx As Double
   Dim riskx As Double
   Dim ok As Boolean


   For i = LBound(L, 1) To UBound(L, 1)
         ok = oml.SolveLP(L(i, 1))
         If (Not ok) Then
           Exit Sub
         End If
         If (n = 0) Then
            n = oml.getNumInstruments()
            ReDim x(n)
         End If
         Call oml.getSolution(x, retx, riskx)
         Range("Solution!P10").Cells(i, 1) = retx
         Range("Solution!P10").Cells(i, 2) = riskx
         For j = 0 To n - 1
            Range("Solution!P10").Cells(i, j + 3) = x(j)
         Next j

   Next i

   Dim xname() As String
   ReDim xname(n)
   For i = 0 To n - 1
      xname(i) = oml.xname(i)
      Range("Solution!P10").Cells(0, i + 3) = xname(i)
   Next i


End Sub
```

The resulting display in the spreadsheet can look like:



As can be seen we solve here 17 QP's or 17 LP's. The optimal portfolios are similar whether we use the QP- or LP-formulation. In this case we used a little bit of Excel VBA code in combination with a .NET DLL. A different approach that could be used to implement this is VSTO.

5.13 COLUMN GENERATION FOR CUTTING STOCK PROBLEMS

This section demonstrates how we can call an algorithm coded in C# from Excel. In this case we use VSTO: Visual Studio Tools for Office.

5.13.1 DELAYED COLUMN GENERATION

The cutting stock problem can be described as follows. Rolls of paper of a given width have to be cut in smaller sized rolls according to a given demand. A good description can be found at Wikipedia: http://en.wikipedia.org/wiki/Cutting_stock_problem.

An elegant algorithm that gives close to optimal solutions (or optimal in many cases) is column generation. This algorithm does not require calculating all possible cutting patterns in advance, but finds interesting patterns to be added to the model automatically. This scheme finds the optimal relaxed solution. Finally we can solve the resulting MIP model. This will give us a solution that will be close to globally optimal. (If the gap between the relaxed solution and the optimal solution of the resulting MIP is less than one we know we found the optimal integer solution).

$a_{i,j}$ integer parameter	Data: the number of final rolls with width *i* in cutting pattern *j*.

$x_j \in \{0,1,2,..\}$ decision variable — The number of rolls to cut according to pattern *j*

$\sum_j a_{i,j} x_j \geq d_i$ — Make sure the number of final rolls of width *i* are sufficient to meet demand.

$\min \sum_j x_j$ — Objective: minimize the total number of rolls used.

When we solve the above LP, called the master problem, by relaxing the x variables for a given set of patterns *J*, we can actually determine if there are more interesting patterns by looking at the duals of the demand equation. From linear programming we know a variable or column will be a candidate to enter the basis (i.e. get a nonzero value) if its reduced cost is negative. This is how the Simplex method selects incoming variables. For a new attractive pattern the same thing holds: we want that the reduced cost is negative

$$\sigma_j = c_j - \pi^T A_j < 0$$

Here A_j is the new pattern, π is the vector of duals and c is the cost. The cost coefficients are ones. We can therefore choose the best new pattern as follows:

$\min 1 - \sum_i \pi_i y_i$ — Minimize the reduced cost of new pattern y

$\sum_i w_i y_i \leq R$ — Make sure the new pattern does not exceed the width of the raw materials.

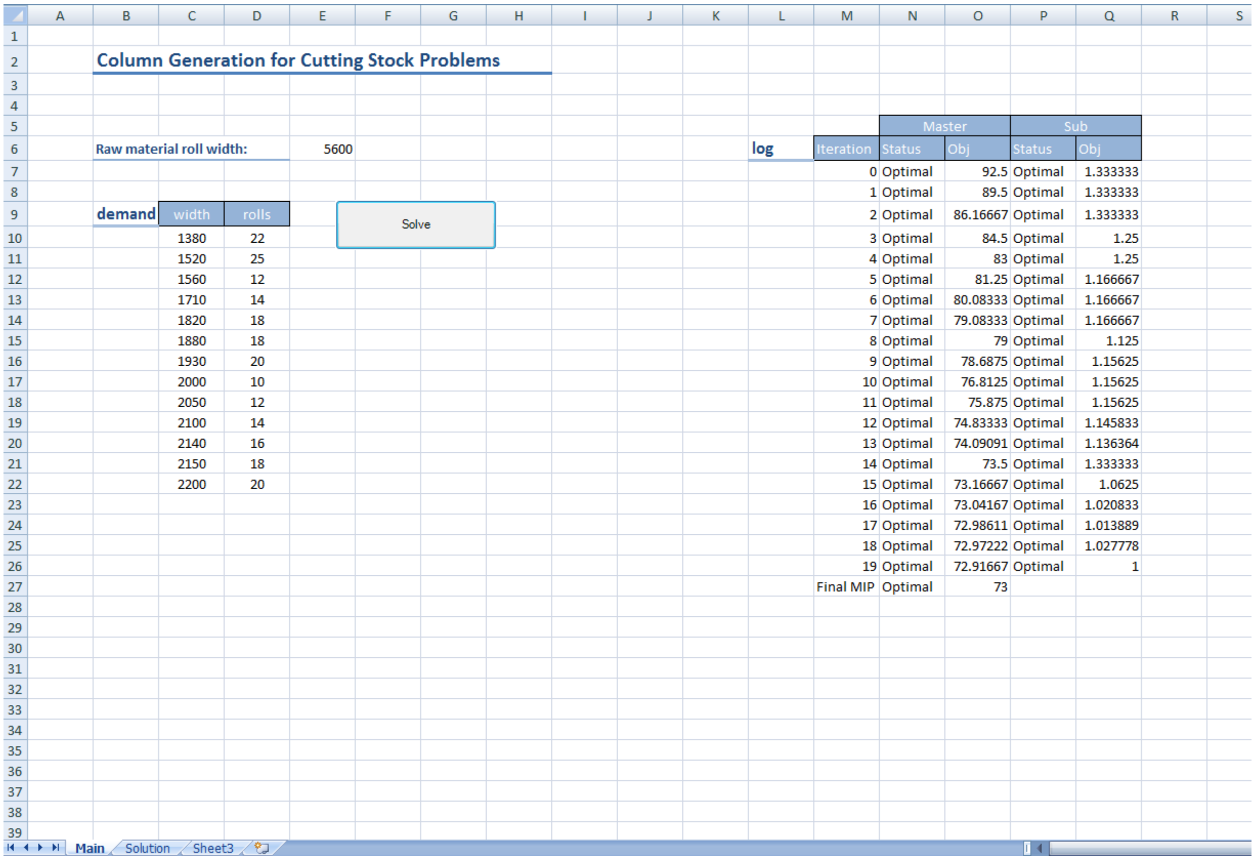
$y_i \in \{0,1,2, \dots\}$

This is called the sub-problem. The sub-problem can be solved as a MIP or using a specialized knapsack solver e.g. based on dynamic programming. Now we can form the complete algorithm:

1. Initialize: form an initial set of patterns
2. Solve the master problem (LP)
3. Solve the sub problem (MIP) using the duals of the master
4. If no new pattern can be found, STOP
5. Add the new pattern to the master problem
6. Go to step 2.

The column generation algorithm is originally from (Gilmore & Gomory, 1961) and (Gilmore & Gomory, 1963). For a simple GAMS implementation see: http://amsterdamoptimization.com/pdf/colgen.pdf.

The VSTO (Visual Studio Tools for Office) application for this problem uses the following input sheet:



Column Generation for Cutting Stock Problems

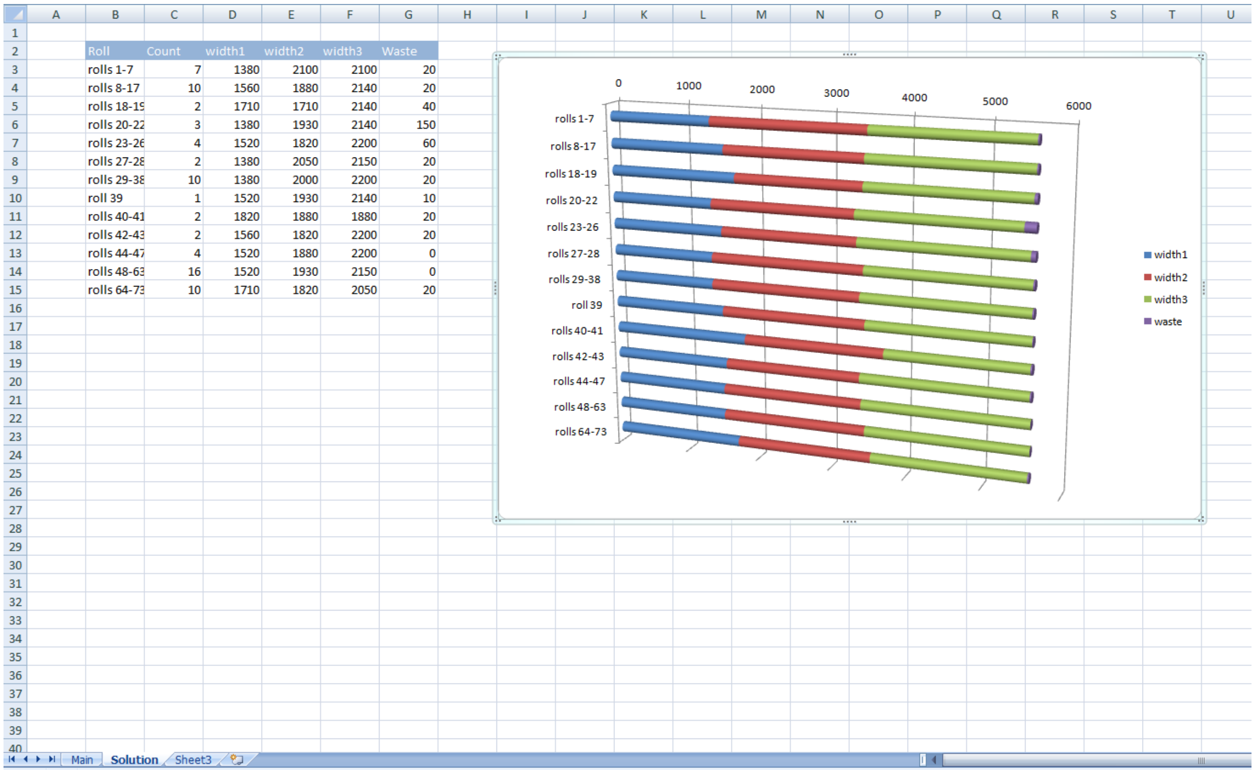
Raw material roll width: 5600

demand	width	rolls
	1380	22
	1520	25
	1560	12
	1710	14
	1820	18
	1880	18
	1930	20
	2000	10
	2050	12
	2100	14
	2140	16
	2150	18
	2200	20

log

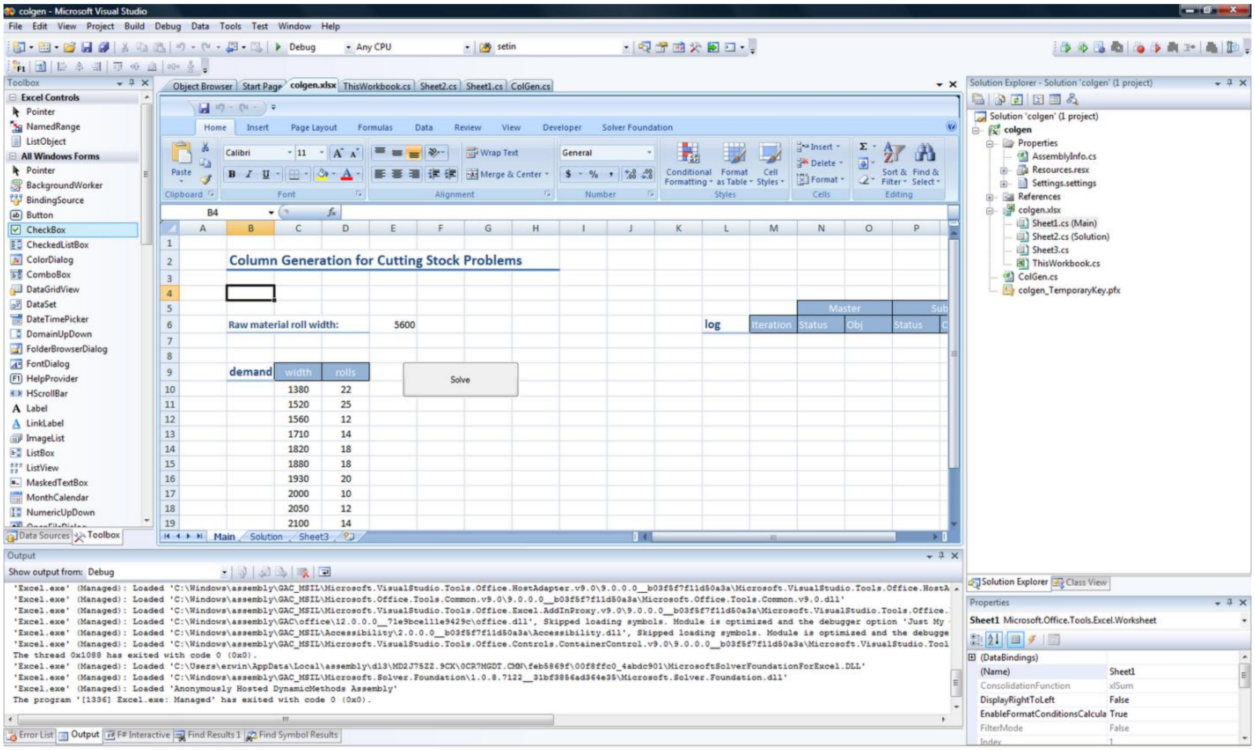
	Master		Sub	
Iteration	Status	Obj	Status	Obj
0	Optimal	92.5	Optimal	1.333333
1	Optimal	89.5	Optimal	1.333333
2	Optimal	86.16667	Optimal	1.333333
3	Optimal	84.5	Optimal	1.25
4	Optimal	83	Optimal	1.25
5	Optimal	81.25	Optimal	1.166667
6	Optimal	80.08333	Optimal	1.166667
7	Optimal	79.08333	Optimal	1.166667
8	Optimal	79	Optimal	1.125
9	Optimal	78.6875	Optimal	1.15625
10	Optimal	76.8125	Optimal	1.15625
11	Optimal	75.875	Optimal	1.15625
12	Optimal	74.83333	Optimal	1.145833
13	Optimal	74.09091	Optimal	1.136364
14	Optimal	73.5	Optimal	1.333333
15	Optimal	73.16667	Optimal	1.0625
16	Optimal	73.04167	Optimal	1.020833
17	Optimal	72.98611	Optimal	1.013889
18	Optimal	72.97222	Optimal	1.027778
19	Optimal	72.91667	Optimal	1
Final MIP	Optimal	73		

The left part is the input. When the button is pressed the progress of the algorithm is shown on the right. The solution is shown in the output sheet:



Roll	Count	width1	width2	width3	Waste
rolls 1-7	7	1380	2100	2100	20
rolls 8-17	10	1560	1880	2140	20
rolls 18-19	2	1710	1710	2140	40
rolls 20-22	3	1380	1930	2140	150
rolls 23-26	4	1520	1820	2200	60
rolls 27-28	2	1380	2050	2150	20
rolls 29-38	10	1380	2000	2200	20
roll 39	1	1520	1930	2140	10
rolls 40-41	2	1820	1880	1880	20
rolls 42-43	2	1560	1820	2200	20
rolls 44-47	4	1520	1880	2200	0
rolls 48-63	16	1520	1930	2150	0
rolls 64-73	10	1710	1820	2050	20

With VSTO we directly code in a .Net language like C#. Excel is integrated nicely in Visual Studio such that development and debugging is easy:



Column Generation for Cutting Stock Problems

Raw material roll width: 5600

demand	width	rolls
	1380	22
	1520	25
	1560	12
	1710	14
	1820	18
	1880	18
	1930	20
	2000	10
	2050	12
	2100	14

log		Master		Sub
	Iteration	Status	Obj	Status

Sheet1 Microsoft.Office.Tools.Excel.Worksheet

(DataBindings)	
(Name)	Sheet1
ConsolidationFunction	xlSum
DisplayRightToLeft	False
EnableFormatConditionsCalcula	True
FilterMode	False

Show output from: Debug

```
'Excel.exe' (Managed): Loaded 'C:\Windows\assembly\GAC_MSIL\Microsoft.VisualStudio.Tools.Office.HostAdapter.v9.0\9.0.0.0__b03f5f7f11d50a3a\Microsoft.VisualStudio.Tools.Office.HostA
'Excel.exe' (Managed): Loaded 'C:\Windows\assembly\GAC_MSIL\Microsoft.Office.Tools.Common.v9.0\9.0.0.0__b03f5f7f11d50a3a\Microsoft.Office.Tools.Common.v9.0.dll'
'Excel.exe' (Managed): Loaded 'C:\Windows\assembly\GAC_MSIL\Microsoft.VisualStudio.Tools.Office.Excel.AddInProxy.v9.0\9.0.0.0__b03f5f7f11d50a3a\Microsoft.VisualStudio.Tools.Office.
'Excel.exe' (Managed): Loaded 'C:\Windows\assembly\GAC\office\12.0.0.0__71e9bce111e9429c\office.dll', Skipped loading symbols. Module is optimized and the debugger option 'Just My
'Excel.exe' (Managed): Loaded 'C:\Windows\assembly\GAC_MSIL\Accessibility\2.0.0.0__b03f5f7f11d50a3a\Accessibility.dll', Skipped loading symbols. Module is optimized and the debugge
'Excel.exe' (Managed): Loaded 'C:\Windows\assembly\GAC_MSIL\Microsoft.VisualStudio.Tools.Office.Controls.ContainerControl.v9.0\9.0.0.0__b03f5f7f11d50a3a\Microsoft.VisualStudio.Tool
The thread 0x1088 has exited with code 0 (0x0).
'Excel.exe' (Managed): Loaded 'C:\Users\erwin\AppData\Local\assembly\dl3\MD2J75ZZ.9CX\0CR7MGDT.CMN\feb5869f\00f8ffc0_4abdc901\MicrosoftSolverFoundationForExcel.DLL'
'Excel.exe' (Managed): Loaded 'C:\Windows\assembly\GAC_MSIL\Microsoft.Solver.Foundation\1.0.8.7122__31bf3856ad364e35\Microsoft.Solver.Foundation.dll'
'Excel.exe' (Managed): Loaded 'Anonymously Hosted DynamicMethods Assembly'
The program '[1336] Excel.exe: Managed' has exited with code 0 (0x0).
```

In Solver Foundation we need run the models at the Solver API level, as that is the only API that provides duals. This is somewhat cumbersome for complex models. Fortunately, the models we need to implement here are very simple. The complete algorithm looks like:

```
using System;
using System.Collections.Generic;
using Microsoft.SolverFoundation.Common;
using Microsoft.SolverFoundation.Solvers;
using Microsoft.SolverFoundation.Services;
using System.Diagnostics;
using SolverFoundation.Plugin.Gurobi;
using Microsoft.VisualStudio.Tools.Applications.Runtime;
using Excel = Microsoft.Office.Interop.Excel;
using Office = Microsoft.Office.Core;



namespace colgen
{
    /// <summary>
    /// log to excel
    /// </summary>
    public class Logger
    {
        /// <summary>
        /// range name of first cell in reporting area
        /// </summary>
        const string rngname = "log";

        /// <summary>
        /// current index
        /// </summary>
        int k = 0;

        /// <summary>
        /// Excel application object
        /// </summary>
        Excel.Application app;

        /// <summary>
        /// constructor
        /// </summary>
        /// <param name="appl">Excel Application</param>
        public Logger(Excel.Application appl)
        {
            app = appl;
            Clear();
        }

        /// <summary>
        /// clear the log
        /// </summary>
        private void Clear()
        {
            Excel.Range r = (Excel.Range)app.get_Range("logx", Type.Missing);
            r.Clear();

        }

        /// <summary>
        /// log the string
        /// </summary>
        /// <param name="s"></param>
        public void log(string s, int j)
        {
            if (j == 1) ++k;
            Excel.Range r = (Excel.Range)app.get_Range(rngname, Type.Missing).Cells.get_Item(k,
j);
            r.set_Value(Excel.XlRangeValueDataType.xlRangeValueDefault, s);
        }
    }


    /// <summary>
    /// Holds the problem data
    /// </summary>
    public class ColGenData
    {
        /// <summary>
        /// raw material roll width
        /// </summary>
        public double rollwidth;

        /// <summary>
        /// Holds the order data
        /// </summary>
        public SortedList<double, int> OrderData;

        /// <summary>
        /// Number of items in OrderData
        /// </summary>
        public int Ndemand;

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="RollWidth"></param>
        /// <param name="DemandWidth"></param>
        /// <param name="DemandCount"></param>
        public ColGenData(double RollWidth, double[] DemandWidth, int[] DemandCount)
        {
            rollwidth = RollWidth;

            Ndemand = DemandWidth.Length;
            if (DemandCount.Length != Ndemand)
                throw new Exception("Arrays DemandWidth and DemandCount do not have the same
length");
            OrderData = new SortedList<double, int>(Ndemand);

            for (int i = 0; i < Ndemand; ++i)
            {
                if (OrderData.ContainsKey(DemandWidth[i]))
                    throw new Exception("DemandWidth array has duplicates");

                OrderData.Add(DemandWidth[i], DemandCount[i]);
            }

        }

        /// <summary>
        /// Get width of order i
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        public double GetWidth(int i)
        {
            return OrderData.Keys[i];
        }

        /// <summary>
        /// Get count of order i
        /// </summary>
        /// <param name="i"></param>
        public int GetCount(int i)
        {
            return OrderData.Values[i];
        }
    }
    /// <summary>
    /// column generation algorithm
    /// </summary>
    public class ColGen
    {

        /// <summary>
        /// Max number of cycles
        /// </summary>
        const int MaxIterations = 100;

        /// <summary>
        /// LP solver. At this level we have access to duals.
        /// </summary>
        ///
        private SimplexSolver lpsolver;

        /// <summary>
        /// mip solver. To solve the knapsack problems.
        /// </summary>
        private SimplexSolver mipsolver;

        /// <summary>
        /// Holds the problem data
        /// </summary>
        public ColGenData Data;

        /// <summary>
        /// Current column in the column generation algorithm (index in lpsolver)
        /// </summary>
        private int currentColumn = -1;


        /// <summary>
        /// Widths used in knapsack problem
        /// </summary>
        public double[] w;

        /// <summary>
        /// Parameters for LP solver
        /// </summary>
        SimplexSolverParams LpParams;

        /// <summary>
        /// Parameters for MIP solver
        /// </summary>
        SimplexSolverParams MipParams;

        /// <summary>
        /// Logging
        /// </summary>
        Logger logger;

        /// <summary>
        /// True if the subproblem found a new pattern
        /// </summary>
        Boolean HaveNewPattern;

        int[] NewPattern;

        /// <summary>
        /// Constructor
        /// </summary>
        public ColGen(double RollWidth, double[] DemandWidth, int[] DemandCount,
Excel.Application appl)
        {
            logger = new Logger(appl);

            Data = new ColGenData(RollWidth, DemandWidth, DemandCount);
            w = new double[Data.Ndemand];
            for (int i = 0; i < Data.Ndemand; ++i)
                w[i] = Data.GetWidth(i);

            NewPattern = new int[Data.Ndemand];

            logger = new Logger(appl);

        }

        /// <summary>
        /// Solve the problem
        /// </summary>
        public void Solve()
        {
            Init();

            for (int Iteration = 0; ; ++Iteration)
            {
                logger.log(string.Format("{0}",Iteration),1);
                Trace.Assert(Iteration < MaxIterations, "Too many iterations");

                SolveMasterProblem();

                SolveSubProblem();

                if (HaveNewPattern)
                    AddColumn(NewPattern);
                else
                    break;

            }
            logger.log("Final MIP", 1);
            SolveMasterAsMip();

            ReportSolution();
        }

        /// <summary>
        /// Report solution back to excel
        /// </summary>
        private void ReportSolution()
        {
            int k1 = 1;
            int k = -1;

            double[] waste = new double[currentColumn-Data.Ndemand];
            int maxnw = 0;

            for (int j = Data.Ndemand + 1; j <= currentColumn; ++j) {
                // number of rolls with this pattern
                int np = (int) Math.Round((double)lpsolver.GetValue(j));
                if (np == 0) continue;
                ++k;
                Globals.Sheet2.SetValue(3 + k, 3, np.ToString());
                // count
                int k2 = k1 + np - 1;
                if (np==1)
                    Globals.Sheet2.SetValue(3 + k, 2, string.Format("roll {0}", k1));
                else
                    Globals.Sheet2.SetValue(3 + k, 2, string.Format("rolls {0}-{1}", k1,k2));
                k1 = k2 + 1;
                // go through column and print the widths
                int nw = 0;
                double usedw = 0;
                for (int i = 0; i < Data.Ndemand; ++i)
                {
                    int nij = (int)lpsolver.GetCoefficient(i + 1, j);
                    if (nij==0) continue;
                    double w = Data.GetWidth(i);
                    for (int ij = 0; ij < nij; ++ij)
                    {
                        ++nw;
                        Globals.Sheet2.SetValue(3 + k, 3 + nw, w.ToString());
                        usedw += w;
                    }

                }
                waste[k] = Data.rollwidth - usedw;
                maxnw = Math.Max(maxnw, nw);
            }

            // print column with waste
            for (int j = 0; j <= k; ++j)
                Globals.Sheet2.SetValue(3 + j, 3 + maxnw + 1, waste[j].ToString());

            // print headers
            const string headerStyle = "60% - Accent1";
            Globals.Sheet2.SetValue(2, 2, "Roll", headerStyle);
            Globals.Sheet2.SetValue(2, 3, "Count", headerStyle);
            for (int i = 0; i < maxnw; ++i)
                Globals.Sheet2.SetValue(2, 4 + i, string.Format("width{0}", i + 1), headerStyle);
            Globals.Sheet2.SetValue(2, 4 + maxnw, "Waste", headerStyle);

            // add graph
            Microsoft.Office.Tools.Excel.Chart ch = Globals.Sheet2.addChart();
            for (int i = 0; i < maxnw; ++i)
            {
                Excel.Series s = 
((Excel.SeriesCollection)ch.SeriesCollection(Type.Missing)).NewSeries();
                s.Name = string.Format("width{0}", i + 1);
                s.Values = string.Format("=Solution!{0}:{1}",
                          Globals.Sheet2.AddressOf(3,4+i),
                          Globals.Sheet2.AddressOf(3+k,4+i));
            }
            Excel.Series s2 = 
((Excel.SeriesCollection)ch.SeriesCollection(Type.Missing)).NewSeries();
            s2.Name = "waste";
            s2.Values = string.Format("=Solution!{0}:{1}",
                      Globals.Sheet2.AddressOf(3, 4 + maxnw),
                      Globals.Sheet2.AddressOf(3 + k, 4 + maxnw));
            s2.XValues = string.Format("=Solution!{0}:{1}",
                      Globals.Sheet2.AddressOf(3, 2),
                      Globals.Sheet2.AddressOf(3 + k, 2));
            ((Excel.Axis) ch.Axes(Excel.XlAxisType.xlCategory, 
Excel.XlAxisGroup.xlPrimary)).ReversePlotOrder = true;
        }

        /// <summary>
        /// Solve master problem: this is an LP
        /// </summary>
        private void SolveMasterProblem()
        {
            ILinearSolution r = lpsolver.Solve(LpParams);
            Trace.Assert(lpsolver.Result == LinearResult.Optimal, "Master is not optimal");
            logger.log(string.Format("{0}", lpsolver.Result.ToString()), 2);
            logger.log(string.Format("{0}", (double)r.GetSolutionValue(0)), 3);
        }

        /// <summary>
        /// Final step
        /// </summary>
        private void SolveMasterAsMip()
        {
            for (int i = Data.Ndemand + 1; i <= currentColumn; ++i)
                lpsolver.SetIntegrality(i, true);
            ILinearSolution r = lpsolver.Solve(MipParams);
            Trace.Assert(lpsolver.Result == LinearResult.Optimal, "Master is not optimal");
            logger.log(string.Format("{0}", lpsolver.Result.ToString()), 2);
            logger.log(string.Format("{0}", (double)r.GetSolutionValue(0)), 3);
        }
        /// <summary>
        /// Solve sub problem: this is a knapsack problem
        /// </summary>
        private void SolveSubProblem()
        {
            //
            // get duals, they become obj coefficients
            //
            ILinearSolverSensitivityReport r =
lpsolver.GetReport(LinearSolverReportType.Sensitivity) as ILinearSolverSensitivityReport;
            Trace.Assert(r != null, "No duals available");
            for (int i = 0; i < Data.Ndemand; ++i)
                mipsolver.SetCoefficient(0, 2 + i, (double)r.GetDualValue(i + 1));


            //
            // solve min 1 - c'y, s.t. w'y <= r  <=> max c'y s.t. w'y <= r
            //
            mipsolver.Solve(MipParams);
            Trace.Assert(mipsolver.Result == LinearResult.Optimal, "Sub is not optimal");
            logger.log(string.Format("{0}", mipsolver.Result.ToString()),4);
            logger.log(string.Format("{0}", (double)mipsolver.GetSolutionValue(0)), 5);


            double obj = (double)mipsolver.GetSolutionValue(0);
            HaveNewPattern = false;
            if (obj > 1.001)
            {
                HaveNewPattern = true;
                for (int i = 0; i < Data.Ndemand; ++i)
                    NewPattern[i] = (int) mipsolver.GetValue(2 + i); // neeed round here?
            }

        }

        /// <summary>
        /// Initialization
        /// </summary>
        private void Init()
        {
            ClearSolutionSheet();

            CreateMasterProblem();

            CreateSubProblem();

            InitialSolution();
        }

        /// <summary>
        /// Clear sheet
        /// </summary>
        private void ClearSolutionSheet()
        {
            Globals.Sheet2.ClearSheet();
        }

        /// <summary>
        /// Create knapsack problem
        /// </summary>
        private void CreateSubProblem()
        {
            //
            // knapsack problem with Ndemand variables
            //
            mipsolver = new SimplexSolver();

            //
            // objective
            //
            int rownum;
            Boolean ok = mipsolver.AddRow("obj", out rownum);
            Trace.Assert(ok, "AddRow returned false");
            Trace.Assert(rownum == 0, "AddRow: Objective should have row number 0");
            mipsolver.AddGoal(0, 1, false);

            //
            // knapsack constraint
            //
            ok = mipsolver.AddRow("knapsack", out rownum);
            Trace.Assert(ok, "AddRow returned false");
            Trace.Assert(rownum == 1, "Unexpected row number from AddRow");
            mipsolver.SetBounds(1, Rational.NegativeInfinity, Data.rollwidth);

            //
            // variables
            //
            for (int j = 0; j < Data.Ndemand; ++j)
            {
                int colnum;
                string s = string.Format("y{0}", j);
                ok = mipsolver.AddVariable(s, out colnum);
                Trace.Assert(ok, "AddVariable returned false");
                mipsolver.SetLowerBound(colnum, 0);
                mipsolver.SetIntegrality(colnum, true);

                //
                // coefficients of knapsack constraint stay fixed
                //
                mipsolver.SetCoefficient(1, colnum, w[j]);
            }


        }

        /// <summary>
        /// Initial empty constraints
        /// </summary>
        private void CreateMasterProblem()
        {

            lpsolver = new SimplexSolver();

            //
            // add obj row as 0-th row
            //
            int rownum;
            Boolean ok=lpsolver.AddRow("obj", out rownum);
            Trace.Assert(ok, "AddRow returned false");
            Trace.Assert(rownum == 0, "AddRow: Objective should have row number 0");
            lpsolver.AddGoal(0, 1, true);


            //
            // the demand equations are numbered 1..NDEMAND
            //  They are all of the form >= demand[i]
            //
            for (int i = 1; i <= Data.Ndemand; ++i)
            {
                string s = string.Format("demand{0}",i);
                ok = lpsolver.AddRow(s, out rownum);
                Trace.Assert(ok, "AddRow returned false");
                Trace.Assert(rownum == i, "Unexpected row number from AddRow");
                lpsolver.SetBounds(i, Data.GetCount(i-1), Rational.PositiveInfinity);
            }

            //
            // column numbers start here.
            //
            currentColumn = Data.Ndemand;

            //
            // we need duals
            //
            LpParams = new SimplexSolverParams();
            LpParams.Algorithm = SimplexAlgorithmKind.Dual;
            LpParams.GetSensitivityReport = true;
            LpParams.PresolveLevel = 0;

            MipParams = new SimplexSolverParams();
        }

        /// <summary>
        /// Add column according to new pattern
        /// </summary>
        /// <param name="p">New pattern</param>
        void AddColumn(int[] p)
        {
            ++currentColumn;
            string s = string.Format("pattern{0}", currentColumn);
            int j;
            Boolean ok = lpsolver.AddVariable(s,out j);
            Trace.Assert(ok,"AddVariable returned false");
            Trace.Assert(currentColumn == j,"Unexpected col number from AddVariable");
            lpsolver.SetLowerBound(j, 0);
            for (int i = 0; i < p.Length; ++i)
                if (p[i]>0)
                {
                    lpsolver.SetCoefficient(i+1,currentColumn,p[i]);
                    lpsolver.SetCoefficient(0, currentColumn, 1);
                }
        }

        /// <summary>
        /// Find an initial set of patterns
        /// We could use here a better algorithm such as a best fit heuristic.
        /// </summary>
        void InitialSolution()
        {
            int[] p = new int[Data.Ndemand];
            for (int i = 0; i < Data.Ndemand; ++i)
            {
                double w = Data.GetWidth(i);
                p[i] = (int) (Data.rollwidth / w);
                AddColumn(p);
                p[i] = 0;
            }
        }

    }
}
```

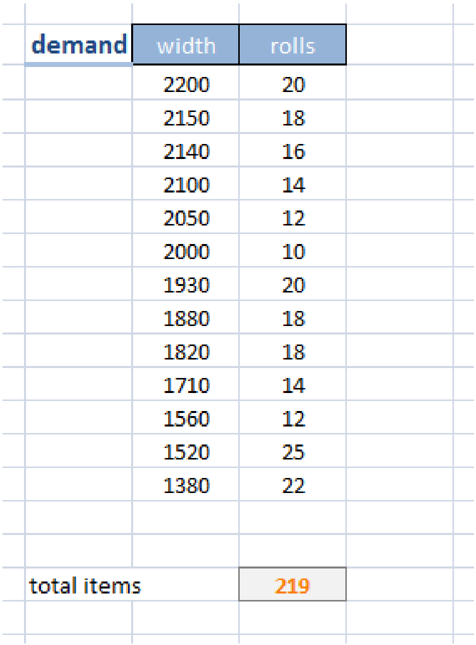
Note that for the LP to create correct duals we did:

```
//
// we need duals
//
LpParams = new SimplexSolverParams();
LpParams.Algorithm = SimplexAlgorithmKind.Dual;
LpParams.GetSensitivityReport = true;
LpParams.PresolveLevel = 0;
```

Resetting the presolve level to zero is related to a bug in the LP solver, which is fixed in the next version.

5.13.2 BIN BACKING FORMULATION

Sometimes a different approach mentioned based on a bin-packing modeling paradigm. Instead of talking about groups of orders with the same width we now focus on individual items. There are 219 items in this example:



demand	width	rolls
	2200	20
	2150	18
	2140	16
	2100	14
	2050	12
	2000	10
	1930	20
	1880	18
	1820	18
	1710	14
	1560	12
	1520	25
	1380	22
total items		219

$i \in \{1, \dots, N\}$ Items to be placed in bins. In the above example we have N=219.

$j \in \{1, \dots, B\}$ Number of bins. We don't know the number in advance, but we need to provide an upper bound. A heuristic may provide this number. From the previous section we know the optimal objective is 73, so let's use B=100 for this experiment.

variable $x_{i,j} \in \{0,1\}$ Binary variable indicating whether item *i* goes into bin *j*. The disadvantage of this formulation is that we have a lot of these. There are $219 \times 100 = 21{,}900$ binary variables.

variable $y_j \in \{0,1\}$ This variable indicates whether bin *j* is used. This variable is automatically integer so we can relax it to $y_j \in [0,1]$. This can make the model marginally more efficient to solve (the number of variables in *y* is small compared to *x*).

$\sum_j x_{i,j} = 1 \ \forall i$ Each item has to be assigned to exactly one bin.

$\sum_i w_i x_{i,j} \le b_j \ \ \forall j$ We cannot exceed the capacity of bin *j*. Here w_i is the width of each item and b_j is width of each bin. For our example all b_j are the same.

$y_j \ge x_{i,j} \ \ \forall i, j$ This models the implication $x_{i,j} = 1 \Rightarrow y_j = 1$. We don't worry about the $\Leftarrow$ part as this is handled automatically by minimizing the number of bins used.

$$\text{minimize} \sum_j y_j$$

The objective is to minimize the number of bins used.

This model can be straightforwardly implemented in OML:

```
Model[

    Parameters[Sets,Items,Bins],
    Parameters[Integers,OrderWidth[Items],BinWidth[Bins]],

    Decisions[Integers[0,1],x[Items,Bins]],
    Decisions[Integers[0,1],y[Bins]],

    Constraints[
        Foreach[{i,Items},Sum[{j,Bins},  x[i,j]]==1 ],
        Foreach[{j,Bins}, Sum[{i,Items}, OrderWidth[i]*x[i,j]] <= BinWidth[j]],
        Foreach[{i,Items},{j,Bins}, y[j] >= x[i,j]]
    ],

    Goals[Minimize[UsedBins->Sum[{j,Bins},y[j]]]]

]
```

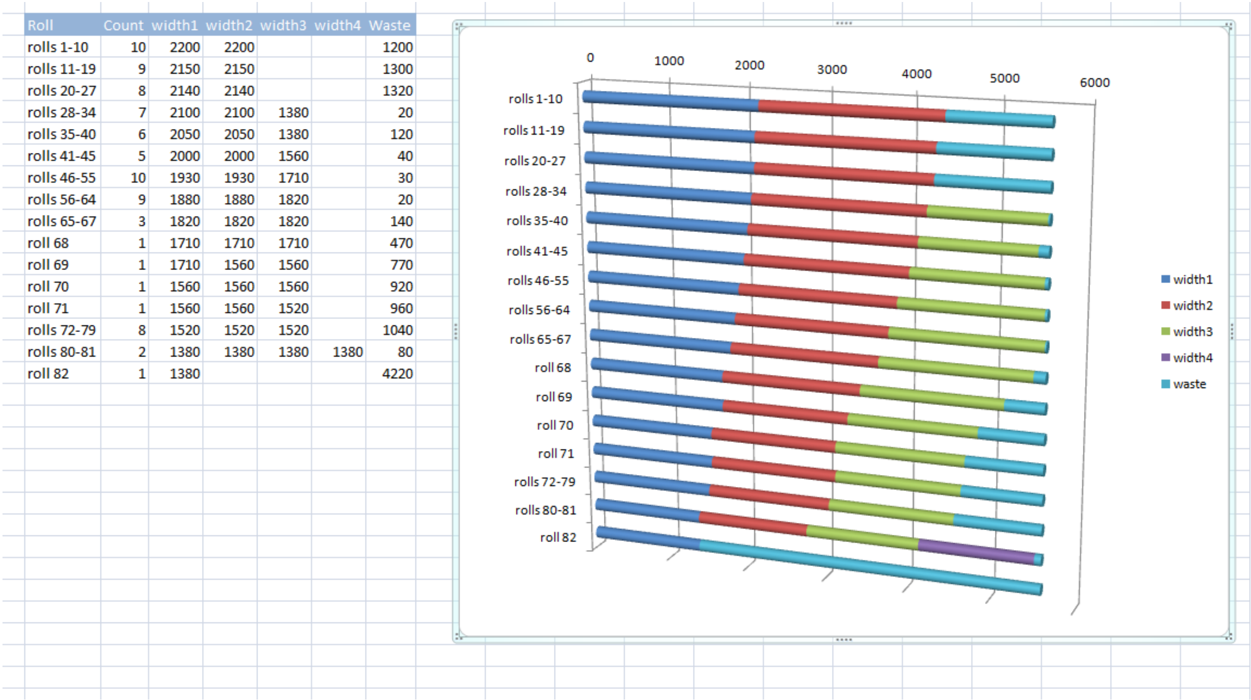
As this model becomes large quickly, it does not solve as fast as the column generation algorithm. For the given data set, using N=100 bines, we end up with model with 22,000 variables (21,900 of which are binary) and 22,220 equations. This is very large even for the best MIP solvers.

5.13.3 HEURISTICS

A well-known heuristic to solve the bin-packing problem is “best-fit decreasing”:

1. Sort the items by size from large to small
2. Insert items in the bin with smallest remaining space where it fits

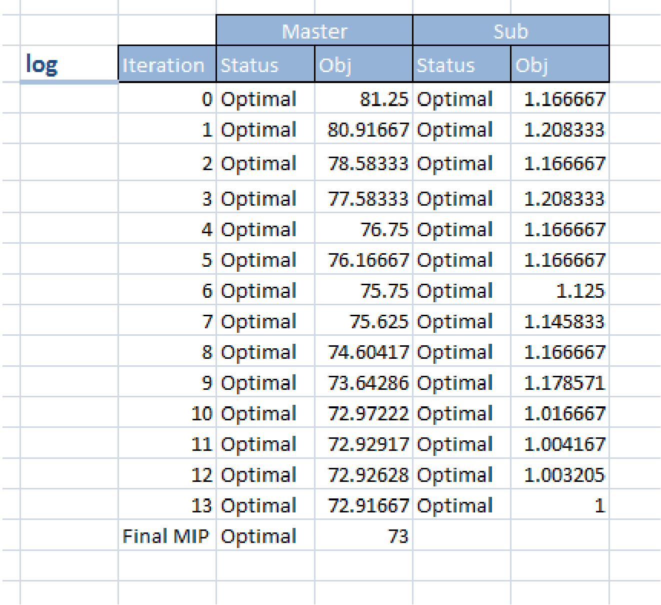
This can be easily coded in VBA. Even a fairly simple minded implementation will be very fast, but the solution found by this algorithm is quite sub-optimal:



Roll	Count	width1	width2	width3	width4	Waste
rolls 1-10	10	2200	2200			1200
rolls 11-19	9	2150	2150			1300
rolls 20-27	8	2140	2140			1320
rolls 28-34	7	2100	2100	1380		20
rolls 35-40	6	2050	2050	1380		120
rolls 41-45	5	2000	2000	1560		40
rolls 46-55	10	1930	1930	1710		30
rolls 56-64	9	1880	1880	1820		20
rolls 65-67	3	1820	1820	1820		140
roll 68	1	1710	1710	1710		470
roll 69	1	1710	1560	1560		770
roll 70	1	1560	1560	1560		920
roll 71	1	1560	1560	1520		960
rolls 72-79	8	1520	1520	1520		1040
rolls 80-81	2	1380	1380	1380	1380	80
roll 82	1	1380				4220

Remember the optimal solution has just 73 rolls.

This heuristic does not give very good results for this data set, but we can still use it to improve the performance of the column generation algorithm. In that algorithm we needed an initial solution. I used before a set of simple patterns: a pattern is just *n* times width *w*, where *n* is the maximum that fits in a roll. We can use basically any feasible solution as starting solution, so using the results of the heuristic make sense. When we run that version, the algorithm stops a little bit quicker:



		Master		Sub	
log	Iteration	Status	Obj	Status	Obj
	0	Optimal	81.25	Optimal	1.166667
	1	Optimal	80.91667	Optimal	1.208333
	2	Optimal	78.58333	Optimal	1.166667
	3	Optimal	77.58333	Optimal	1.208333
	4	Optimal	76.75	Optimal	1.166667
	5	Optimal	76.16667	Optimal	1.166667
	6	Optimal	75.75	Optimal	1.125
	7	Optimal	75.625	Optimal	1.145833
	8	Optimal	74.60417	Optimal	1.166667
	9	Optimal	73.64286	Optimal	1.178571
	10	Optimal	72.97222	Optimal	1.016667
	11	Optimal	72.92917	Optimal	1.004167
	12	Optimal	72.92628	Optimal	1.003205
	13	Optimal	72.91667	Optimal	1
	Final MIP	Optimal	73		

We see that in iteration 0 we start now with a better Master: the objective is 81.25 instead of 92.5 with the simple initial solution. The total number of major iterations is reduced from 19 to 13.

6 BIBLIOGRAPHY

Carlier, J., & Pinson, E. (1989). An algorithm for solving the job shop problem. *Management Science , 35* (2), 164-176.

Dantzig, G. B. (1963). *Linear Programming and Extensions.* Princeton: Princeton University Press.

Fisher, H., & Thompson, G. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In G. T. J.F. Muth, *Industrial Scheduling.* Englewood Cliffs, New Jersey: Prentice Hall.

Fourer, R. (1998). Extending a General-Purpose Algebraic Modeling Language to Combinatorial Optimization: A Logic Programming Approach. In D. Woodruff, *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search: Interfaces in Computer Science and Operations Research* (pp. 31-74). Dordrecht, The Netherlands : Kluwer Academic Publishers.

Fylstra, D., Lasdon, L., Watson, J., & Warren, A. (1998). Design and Use of the Microsoft Excel Solver. *Interfaces , 28* (5), 25-55.

Gilmore, P. C., & Gomory, R. E. (1963). A linear programming approach to the cutting-stock problem - Part II. *Operations Research , 11*, 863-888.

Gilmore, P. C., & Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research , 9*, 849-859.

Jain, A., & Meeran, S. (1998). Deterministic Job-Shop Scheduling: Past, Present and Future. *European Journal of Operational Research , 113*, 390-434.

Knuth, D. (2004). *A Draft of Section 7.2.1.2: Generating All Permutations.* Retrieved from http://www-cs-faculty.stanford.edu/~knuth/fasc2b.ps.gz.

Manne, A. (1960). On the job-shop scheduling problem. *Operations Research , 8* (2).

Murphy, F. H. (1996). Annotated bibliography on linear programming models. *Interactive transactions on ORMS , 1* (4).

Panko, R. R. (1998). What We Know About Spreadsheet Errors. *Journal of End User Computing , 10* (2), 15-21.

Smith, B. (2000). *Modelling a Permutation Problem.* University of Leeds, School of Computer Studies.

Stigler, G. J. (1945). The cost of subsistence. *Journal of Farm Economics , 27*, 303–314.

Svestka, J. (1978). A continuous variable representation of the TSP. *Mathematical Programming , 15*, 211-213.

van Hentenryck, P. (1999). *The OPL optimization programming language.* Cambridge, MA, USA: MIT Press.

Williams, H. P., & Yan, H. (2001). Representations of the all-different Predicate of Constraint Satisfaction in Integer Programming. *INFORMS Journal on Computing , 13*, 96-103.

Wirth, N. (1971). Program Development by Stepwise Refinement. *Communications of the ACM , 14* (4), 221-227.

[1] Source: http://www.solver.com/pressinfo.htm

[2] Even carefully crafted spreadsheets have an error rate of 1% in their formulas (Panko, 1998).

[3] Actually the data for the model are not identical: the GAMS version introduces degeneracy, i.e. multiple solutions exist with the same optimal objective value.

[4] To be precise, the bounds applied to an integer domain for `Integers[lo,hi]` are `Ceiling[lo]` and `Floor[hi]`.

[5] It would be a useful extension to allow bound data to be used as bounds.