Challenge September - 2018

Balanced Assignment

A solution with OPL by Alex Fleischer afleischer@fr.ibm.com

OPL (Optimization Programming Language) is an abstract modeling language that helps model easily optimization problems that can be solved both with IBM CPLEX linear programming and IBM CPLEX constraint programming CPOptimizer (CPO)

Let us remember that with ILOG (French company bought by IBM in 2009) we had two kind of decision engines:

A) Rule based (JRules, ODM)
B) Optimization based (CPLEX)

Here a small example of a tiny optimization model, in English, OPL and Python

Zoo, bus, kids and optimization

With words	In OPL	In Python / DoCplex

300 kids need to travel to the London zoo
The school may rent 40 seats and 30 seats buses for 500 and 400 £
How many buses of each to minimize cost ?
 
	int nbKids=300;
	float costBus40=500;
	float costBus30=400;
	
	dvar int+ nbBus40;
	dvar int+ nbBus30;
	
	minimize
	costBus40*nbBus40
	+nbBus30*costBus30;
	
	subject to
	{
	40*nbBus40+
	nbBus30*30
	>=nbKids;
	}
		from docplex.mp.model import Model
		
		mdl = Model(name='buses')
		
		nbbus40 =
		mdl.integer_var(name='nbBus40')
		nbbus30 =
		mdl.integer_var(name='nbBus30')
		
		mdl.add_constraint(nbbus40*40 +
		nbbus30*30 >= 300, 'kids')
		
		mdl.minimize(nbbus40*500 +
		nbbus30*400)
		
		mdl.solve()
		
		print(nbbus40.solution_value);
		print(nbbus30.solution_value);

We can call CPLEX from many languages (C,C++,.NET,Java,Python …) but using OPL leads to a clear frontier between the model and the code that will embed the model. (Not far from Decision Model and Notation (DMN) principle : “The notation is designed to be readable by business and IT users alike. This enables various groups to effectively collaborate in defining a decision model”)

Now let’s move to our Balanced Assignment.

A run configuration in OPL has some data and a model.

The data part to read the excel spreadsheet is:

```
SheetConnection s("BalancedAssignment.xls");

participants from SheetRead(s,"A2:E211");

nbGroups=12;

possibleSizeMin=16;
possibleSizeMax=19;
```

and then the model part:

```
// tuple with components for all participants
tuple p
{
key string name;
string c1;
string c2;
string c3;
string c4;
}

// All participants
{p} participants=...;

// properties of a participant
string c[p in participants][j in
1..4]=(j==1)?p.c1:((j==2)?p.c2:((j==3)?p.c3:(p.c4)));

{string} options[j in 1..4]={c[p][j] | p in participants};

tuple rankandoption
{
int j;
string option;
}
```

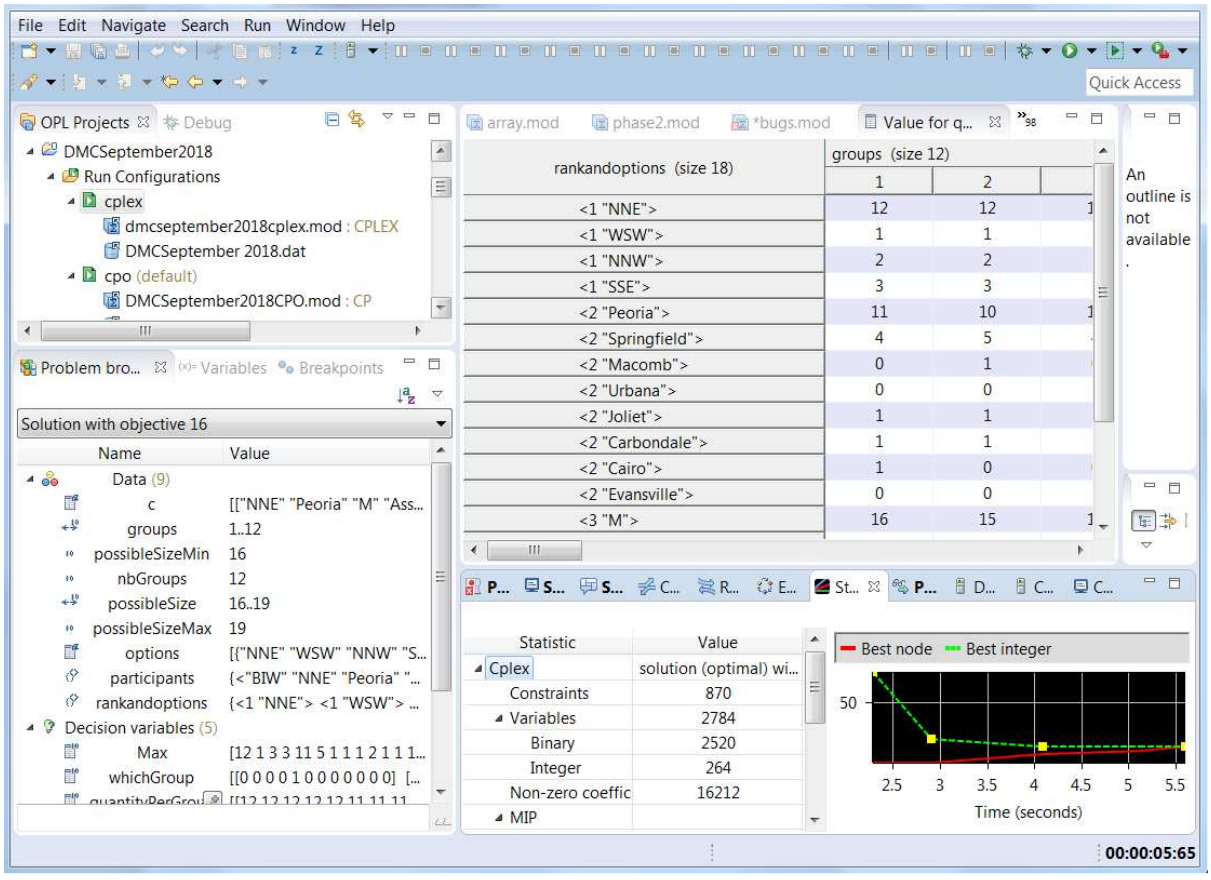
```
{rankandoption} rankandoptions={<j,o> | j in 1..4,o in options[j]};



int nbGroups=...;

range groups=1..nbGroups;

int  possibleSizeMin=...;
int  possibleSizeMax=...;

range possibleSize=possibleSizeMin..possibleSizeMax;

dvar boolean whichGroup[participants][groups];
dvar int groupSize[groups] in possibleSize;
dvar int quantityPerGroup[rankandoptions][groups];
dvar int Max[rankandoptions];
dvar int Min[rankandoptions];

// Minimize max - min of numbers of participants in a group for a property
minimize sum(i in 1..4)

sum(o in options[i])
(Max[<i,o>]-Min[<i,o>]);



subject to
{

// one and only one group per participant
forall(p in participants) sum(g in groups) whichGroup[p][g]==1;

// compute sizes of groups
forall(g in groups) groupSize[g]==sum(p in participants) whichGroup[p][g];

// compute quantities of a given property for all groups
forall(ro in rankandoptions,g in groups) quantityPerGroup[ro,g]==sum(p in
participants:c[p][ro.j]==ro.option)whichGroup[p,g];

// min and max of quantities for all groups
forall(ro in rankandoptions,g in groups) quantityPerGroup[ro,g]>=Min[ro];
forall(ro in rankandoptions,g in groups) quantityPerGroup[ro,g]<=Max[ro];

}

{string} namesPerGroup[g in groups]={p.name | p in participants :
whichGroup[p,g]==1};

execute
{
writeln(namesPerGroup);
}


assert 4*card(participants)==sum(ro in rankandoptions,g in groups)
quantityPerGroup[ro,g];
```

When we run this in the CPLEX IDE, we get the optimal solution in 5 s and the objective is 16

We see

rankandoptions (size 18)	groups (size 12)											
	1	2	3	4	5	6	7	8	9	10	11	12
<1 "NNE">	12	12	12	12	12	11	11	11	11	12	12	12
<1 "WSW">	1	1	1	1	1	1	1	1	1	1	1	1
<1 "NNW">	2	2	3	2	2	2	3	2	3	2	3	2
<1 "SSE">	3	3	2	2	2	3	3	3	3	2	2	2
<2 "Peoria">	11	10	10	10	11	10	11	10	11	10	11	10
<2 "Springfield">	4	5	4	4	4	4	4	4	4	4	4	4
<2 "Macomb">	0	1	0	1	0	1	0	0	0	1	0	1
<2 "Urbana">	0	0	1	0	0	0	0	1	1	0	0	0
<2 "Joliet">	1	1	1	1	0	1	1	1	1	1	0	1
<2 "Carbondale">	1	1	1	1	1	1	2	1	1	1	2	1
<2 "Cairo">	1	0	0	0	0	0	0	0	0	0	1	0
<2 "Evansville">	0	0	1	0	1	0	0	0	0	0	0	0
<3 "M">	16	15	15	15	15	15	15	15	15	15	16	15
<3 "F">	2	3	3	2	2	2	3	2	3	2	2	2
<4 "Assistant">	8	7	8	8	8	7	8	7	7	7	8	7
<4 "Adjunct">	8	8	8	7	7	7	8	7	8	7	8	7
<4 "Deputy">	2	2	2	2	2	2	2	2	2	2	2	2
<4 "Consultant">	0	1	0	0	0	1	0	1	1	1	0	1

The full IDE looks like



NB: We used linear programming and minimized the sum of max – min but we can also use Constraint Programming and minimize max of standard deviation.

To do that we simply turn the model into

```
using CP;

execute
{
// time limit

cp.param.timelimit=60;
}

tuple p
{
key string name;
string c1;
string c2;
string c3;
string c4;
}

{p} participants=...;

// properties of a participant
string c[p in participants][j in
1..4]=(j==1)?p.c1:((j==2)?p.c2:((j==3)?p.c3:(p.c4)));

{string} options[j in 1..4]={c[p][j] | p in participants};

tuple rankandoption
{
int j;
string option;
}

{rankandoption} rankandoptions={<j,o> | j in 1..4,o in options[j]};



int nbGroups=...;

range groups=1..nbGroups;

int  possibleSizeMin=...;
int  possibleSizeMax=...;

range possibleSize=possibleSizeMin..possibleSizeMax;

dvar int whichGroup[participants] in groups;
dvar int groupSize[groups] in possibleSize;
dvar int quantityPerGroup[rankandoptions][groups];

minimize max(i in 1..4)

max(o in options[i])
standardDeviation(all(g in groups)quantityPerGroup[<i,o>][g]);
```

```
subject to


{
// compute sizes of groups
forall(g in groups) groupSize[g]==count(whichGroup,g) ;

// compute quantities of a given property for all groups
forall(ro in rankandoptions,g in groups) quantityPerGroup[ro,g]==count(all(p in
participants:c[p][ro.j]==ro.option)whichGroup[p],g);


}

{string} namesPerGroup[g in groups]={p.name | p in participants :
whichGroup[p]==g};

execute
{
writeln(namesPerGroup);
}


assert 4*card(participants)==sum(ro in rankandoptions,g in groups)
quantityPerGroup[ro,g];
```

Making Decision Optimization Simple : https://www.linkedin.com/pulse/making-decision-optimization-simple-alex-fleischer/