Challenge January - 2020

NIM

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 the Nim challenge. (January 2020 DMC challenge)

In OPL CPLEX no need to be very clever, we need to translate the game.

```
int m=3; // 1,2 or 3
 {int} possibleStarts={15,16,17};

int n=max(k in possibleStarts) k;
range r=1..n;


//the player who removes the last one loses
// remove 1,2 or 3

// is the position a winning position ?
dvar int winning[1..n] in 0..1;

subject to
{
  winning[1]==0; // lose

  // if one option removing 1 to m you get into a lose then you win
  forall(i in 2..n)
       (winning[i]==1) ==
        ( 1<= sum(j in 1..m:i-j>=1) (winning[i-j]==0));

  // if all removing 1 to m lead to win then you lose
  forall(i in 2..n)
       (winning[i]==0) ==
       ((( sum(j in 1..m:i-j>=1) 1)== sum(j in 1..m:i-j>=1) (winning[i-j]==1))) ;
}

// what to do ? How many balls to remove. 0 means if we have this we should let
the opponent play

int take[i in r]=(winning[i]==0)?0:(i-max(j in 1..i:winning[j]==0)j);

assert forall(i in r:winning[i]==1) winning[i-take[i]]==0;
```

```
execute
{
  for(var i in r)
  if (take[i]!=0) writeln("if remaining ",i," then take ",take[i]);
  else writeln("if remaining ",i," then let the opponent play");
}
```

Which gives

```
if remaining 1 then let the opponent play
if remaining 2 then take 1
if remaining 3 then take 2
if remaining 4 then take 3
if remaining 5 then let the opponent play
if remaining 6 then take 1
if remaining 7 then take 2
if remaining 8 then take 3
if remaining 9 then let the opponent play
if remaining 10 then take 1
if remaining 11 then take 2
if remaining 12 then take 3
if remaining 13 then let the opponent play
if remaining 14 then take 1
if remaining 15 then take 2
if remaining 16 then take 3
if remaining 17 then let the opponent play

We could have been clever and proved that if n=4*k+1 with k integer then the
position is lose and build rules out of that. But with OPL CPLEX, no need for that
and the solution could adapt to some change in the game.
```

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