OPL language
and IBM ILOG CPLEX Optimization Studio

Cédric Pralet
ONERA Toulouse

cedric.pralet@onera.fr

Introduction

OPL : Optimization Programming Language.

A high level modeling language to write linear programming, integer linear programming, and constraint programming models

Offers a catalog of simple, powerful, and intuitive expressions

Elements of an OPL model :

1. data ;
2. variables ;
3. criterion ;
4. constraints ;
5. commands for pre or post-processing or for search control (optional).

Example 1 : Linear Programming (LP)

Goal : optimize profit for the production of NH_3 and NH_4Cl.
Stocks : 50 units of N, 180 units of H, 40 units of Cl.
Profits : 40 € per unit of NH_3, 50 € per unit of NH_4Cl.

Model : gaz.mod

```
dvar float+ gas;
dvar float+ chloride;

maximize 40 * gas + 50 * chloride;
constraints {
   gas + chloride <= 50;           // N
   3 * gas + 4 * chloride <= 180; // H
   chloride <= 40;                 // Cl
};
```

Result

```
OBJECTIVE: 2300, gas = 20.0000, chloride = 30.0000
```

Example 2 : Integer Linear Programming (ILP)

File knapsack.mod

```
int Capacity = 20;
int NbItems = 10;
range Items = 1..NbItems;
int Size[Items] = [1,3,4,2,4,3,4,3,3,3];
int Value[Items] = [10,40,10,40,20,20,20,30,30,30];

dvar int take[Items] in 0..1;

maximize sum(i in Items) Value[i] * take[i];

constraints {
      sum(i in Items)  take[i] * Size[i] <= Capacity;
}
```

Result

```
OBJECTIVE: 200, take : [1 1 0 1 0 1 0 1 1 1]
```

Separation between model and data

Possibility to separate model (in a file with "`.mod`" extension) and data (in a file with "`.dat`" extension)

File `genericKnapsack.mod`

```
int Capacity = ...;
int NbItems = ...;
range Items = 1..NbItems;
int Size[Items] = ...;
int Value[Items] = ...;

dvar int take[Items] in 0..1;

maximize sum(i in Items) Value[i] * take[i];

constraints {
      sum(i in Items)  take[i] * Size[i] <= Capacity;
}
```

Separation between model and data

File genericKnapsack.dat

```
Capacity = 20;
NbItems = 10;
Size = [1,3,4,2,4,3,4,3,3,3];
Value = [10,40,10,40,20,20,20,30,30,30];
```

Example 3 : Constraint Programming (CP)

Goal : N queens over a N×N board so that no two queens can attack each other

File queens.mod

```
using CP;
int N = 4; // board size
range Domain = 1..N;
dvar int queens[Domain] in Domain;
constraints {
      forall(i,j in Domain : i < j) {
         queens[i] != queens[j];
         abs(queens[i] - queens[j]) != j-i;
      };
};

execute{
writeln(queens);
}
```

General syntax elements of OPL

Arithmetic operations

+	addition
-	minus
*	times
/	division
div	integer division
mod	modulo
abs	absolute value
^	exponent

Comparisons

==	equal
!=	different
>	greater than
<	less than
>=	greater or equal
<=	less or equal

Logic connectors

&&	logical and
||	logical or
!	negation
=>	implication
==	equivalence

Some OPL expressions

Examples of use of keyword `forall`

```
forall(i in Indices) x[i] != y[i];

forall(i,j in Indices) x[i] != y[j];

forall(i,j in Indices : i < j) x[i] != y[j];

forall(i,j in Indices : i < j){
  x[i] != y[j];
  x[i] + y[j] < 10;
}

forall(i in Indices1, j in Indices2 : 2*i >= j){
  x[i] != y[j];
  x[i] + y[j] < 10;
}
```

Warning : expressions used after "such that" (after " : " in "`forall(... : ...)`") cannot depend on variables, only on data

Some OPL expressions

Constraints using min, max, prod, sum

```
max(i in Indices) x[i] == 10;

min(i in Indices) x[i] == 10;

prod(i in Indices) x[i] == 10;

sum(i in Indices) x[i] <= 10;

sum(i in Indices) (x[i] == 0) <= 2;

sum(i in Indices) (x[i] > 3) != 0;
```

Some OPL expressions

Variables of type dexpr, defined as a function of other variables

```
dvar int x in 0..10;
dvar int y in 0..10;
dexpr int z = x + y; // variable representing x + y
```

```
dvar int x[1..10] in 0..1;
dexpr int z = sum(i in 1..10) x[i];
```

```
dvar int x[1..10] in 0..1;
dvar int y[1..10] in 0..1;
dexpr int z[i in 1..10] = x[i] + y[i];
```

No search on expression values, only on variable values

Variable choice heuristics

Possibility to specify heuristics to try and boost search

Fichier pb.mod

```
int N = 4;
dvar int x[1..N] in 0..10;
dvar int y[1..N] in 0..10;
dvar int z[1..N] in 0..5;
dexpr int t[i in 1..N] = x[i] + y[i];

execute {
  var phase1 = cp.factory.searchPhase(x);
  var phase2 = cp.factory.searchPhase(y);
  cp.setSearchPhases(phase1, phase2);
}

constraints {
  forall(i in 1..N){
        (x[i] = 0) => (z[i] <= y[i]);
        (x[i] != 0) => (t[i] >= 10);
  }
}
```

Search limits

Possibility to specify time limits

Fichier pb1.mod

```
execute {
  var phase1 = cp.factory.searchPhase(x);
  var phase2 = cp.factory.searchPhase(y);
  cp.setSearchPhases(phase1, phase2);
  cp.param.TimeLimit = 60;
}
```

Possibility to specify fail limits

Fichier pb2.mod

```
execute {
  var phase1 = cp.factory.searchPhase(x);
  var phase2 = cp.factory.searchPhase(y);
  cp.setSearchPhases(phase1, phase2);
  cp.param.FailLimit = 100000;
}
```

IBM ILOG CPLEX Optimization Studio

Tool for solving OPL model, using two solving tools :

- CPLEX tool for LP and ILP,
- CP Optimizer tool for CP.

By default, use of *CPLEX* (with some automatic constraint linearizations if needed).

Use of *CP Optimizer* by adding keyword "`using CP;`" at the beginning of the model.

IBM ILOG CPLEX Optimization Studio capabilities

Constraint–based optimization (with a single criterion)	CONTINUOUS VARIABLES	DISCRETE VARIABLES
LINEAR	**LP** Linear Programming; **MIP** Mixed Integer Programming	**ILP** Integer Linear Programming
NON LINEAR	~~**NLP** Non Linear Programming~~	**CP** Constraint Programming

Note : NLP in general not handled, but possibility to use some quadratic expressions

Use of OPL Studio

1. Launch OPL Studio
2. Create a new project : *File* $\rightarrow$ *New* $\rightarrow$ *OPL project*
3. Add a model file (.mod) : *right click on the project name* $\rightarrow$ *New* $\rightarrow$ *Model*
4. Add a data file (.dat) : *right click on the project name* $\rightarrow$ *New* $\rightarrow$ *Data*
5. Creation of a configuration of execution : *right click on the project name* $\rightarrow$ *New* $\rightarrow$ *Configuration of execution*
6. Inclusion of a .mod and a .dat in a configuration of execution using drag and drop
7. Execution of the configuration of execution : *right click on the configuration* $\rightarrow$ *Execute this configuration*