OPL: a modelling language

Carlo Mannino
(from OPL reference manual)

ILOG Optimization Programming Language

- OPL is an “Optimization Programming Language
  - *Easy to generate and solve an lp models (program)*
  - *It also provides all features of a standard programming language*
- *We will use the IDE user interface*

- An OPL model consists of:
- a sequence of declarations
- optional preprocessing instructions
- the model/problem definition
- optional postprocessing instructions

- Model are stored in files with extension *.mod*

Data types, constants and variables

- Defining a model:

  - First you need to define variables and constants (names and types)
  - Basic types: *float, int, string.*

  Ex. (constants)

  int i = 32;

  float+ k = 12.7;

  string s = ‘Optimization’;

  - Variables are introduced by “*dvar*”

    dvar float+ x; (rational non negative variable)

    dvar int y; (integer variable)

A simple model

- LP models *translate* naturally to OPL models

LP model

$$
\begin{aligned}
&\max 40g + 50c \\
&s.t. \\
&g + c \le 5 \\
&3g + 5c \le 18 \\
&g \le 4 \\
&g, c \in R
\end{aligned}
$$

OPL model

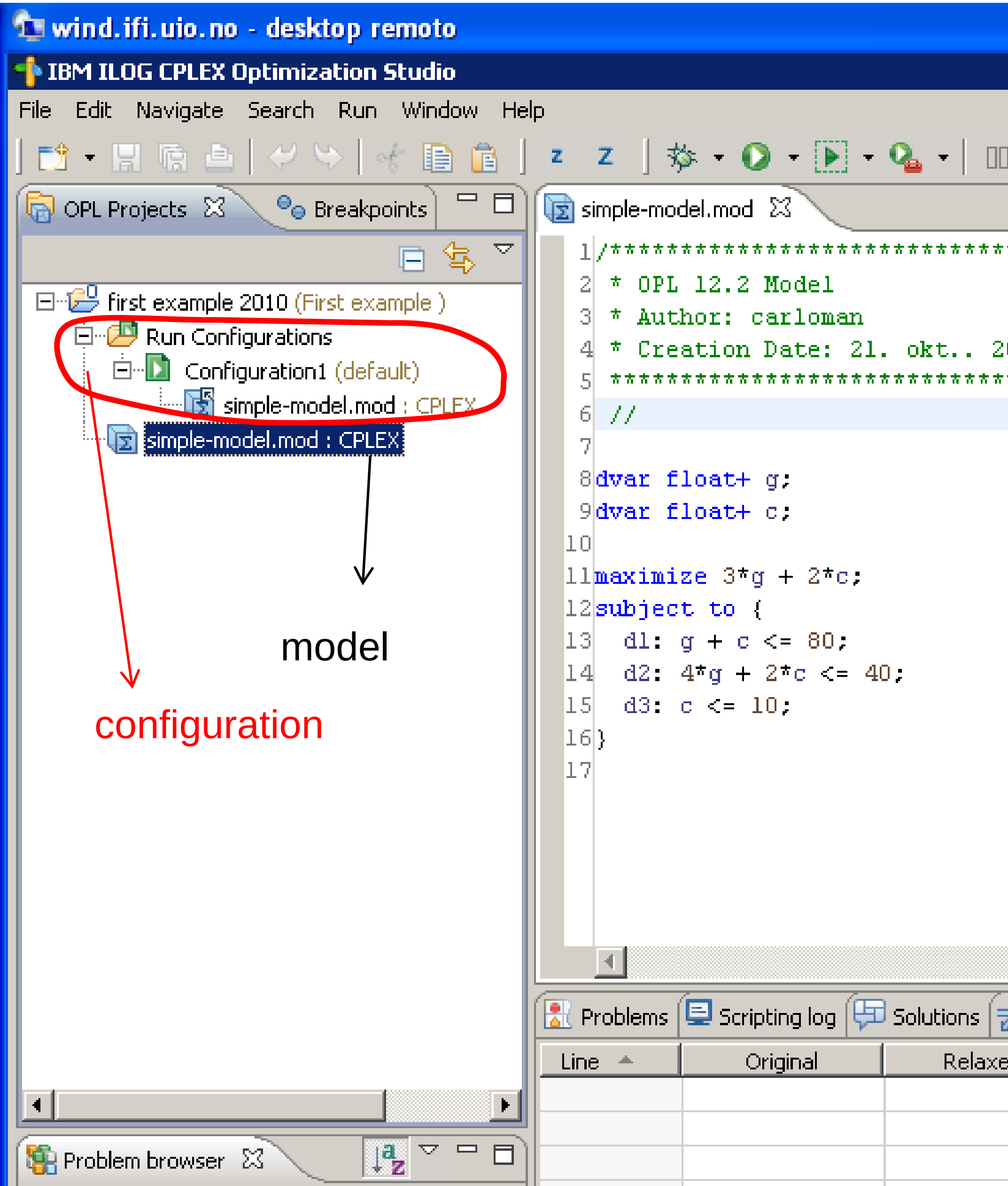
```
dvar float g;
dvar float c;

maximize  40*g + 50*c;
 subject to {
   d1: g + c <=5;
   d2: 3*g + 5*c <= 18;
   d3: g <= 4;
}
```

- OBS: you can associate a name with every constraint by putting it before the constraint, followed by colon ‘:’
- Constraint name = corresponding dual variable (name)

Using OPL interface (IDE)

- To solve models you need (to create) a project.
- Projects can contain several models (and instances).
- To extract a specific model (and a specific instance) you need to define a *configuration*.
- A project can thus contain several configurations, each containing a specific pair *model – instance*.
- Once you create a new configuration, the corresponding model can be imported by drag-and-drop

Using OPL interface (IDE)



Example:

- Project Name: First example 2010
- The project contains only one model (*simple-model.mod*)
- The project contains only one configuration (*Configuration*1)
- Configuration1 contains model *simple-model.mod*

Ranges and arrays

- Ranges of integers are defined as

range Rows = 1..32;

int n = 32;

range Cols = 3..n;

- Ranges are used to define arrays

```
range nodes = 1..5;
range edges = 1..7;
float weight[nodes] = [1, 4, 6.1, 7, 2.2];
float A[nodes][edges] = [
                    [1, 0, 0, 1, 0, 0, -1]
                    [-1, 0, 0, 0, 1, 0, 1]
                              ...
                  ];
```

- Ranges are also used in summations and loops
  - forall (i in items) …

Example: ranges and arrays

$$\max \sum_{i=1}^{n} c_i x_i$$

$$s.t.$$

$$\sum_{i=1}^{n} w_i x_i \le k$$

$$x \in \{0,1\}^n$$

$$c = \begin{pmatrix} 5 \\ 8 \\ 4 \\ 7 \\ 9 \end{pmatrix} \qquad w = \begin{pmatrix} 1 \\ 5 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

```
int n = 5;
range Items = 1..n;
int w[Items] = [1, 5, 2 , 3, 4];
int c[Items] = [5, 8, 4,  7, 9];
int k = 8;
dvar int x[Items] in 0..1;

maximize   sum(i in Items) c[i]*x[i];
subject to {
  knap: sum(i in Items) w[i]*x[i] <= k;
}
```

- Ranges are used in summations and loops
  - forall (i in items) …

Separating models and data

- A model is something different from *one of its instances*
- The linear program for the shortest path problem has a structure which is the same for any graph and any weight function.
- We only need one model, and then we can apply the model to any instance of the shortest path problem
- OPL allows to maintain a strict separation between the model and its instances.
- Models are stored in xxx.mod files
- Instances are stored in yyy.dat files
- In the standard IDE interface, xxx can be different from yyy
- The “Configuration” matches the model with the instance.

Separating models and data

$$\max \sum_{i=1}^{n} c_i x_i$$

$$s.t. \quad \sum_{i=1}^{n} w_i x_i \le k$$

$$x \in \{0,1\}^n$$

$n = 5$ $\quad c = \begin{pmatrix} 5 \\ 8 \\ 4 \\ 7 \\ 9 \end{pmatrix}$ $\quad w = \begin{pmatrix} 1 \\ 5 \\ 2 \\ 3 \\ 4 \end{pmatrix}$ $\quad k = 8$

Knapsack.mod

```
int n = …;
range Items = 1..n;
int w[Items] = …;
int c[Items] = …;
int k = …;
dvar int x[Items] in 0..1;

maximize   sum(i in Items) c[i]*x[i];
subject to {
  knap: sum(i in Items) w[i]*x[i] <= k;
}
```

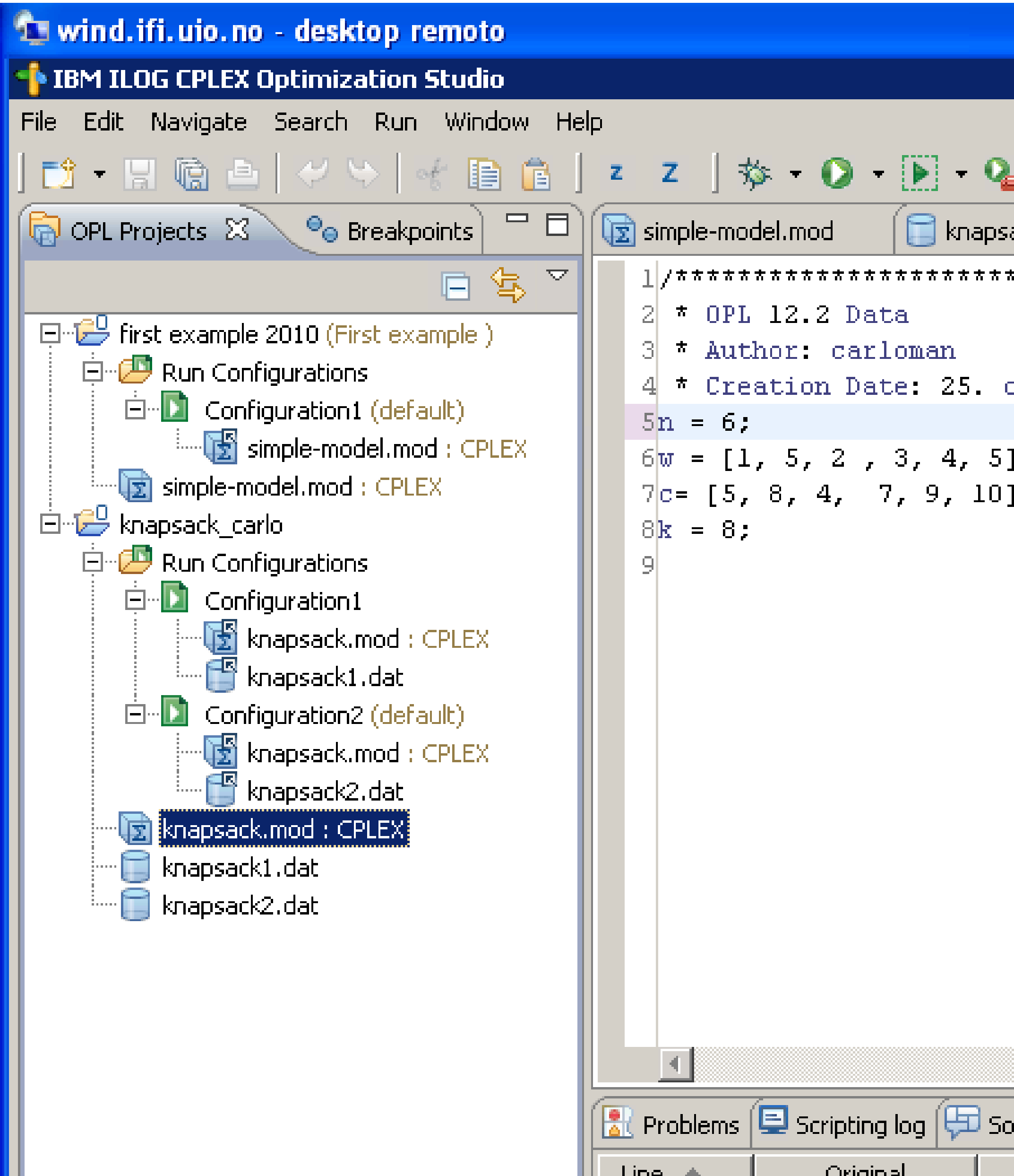
Knapsack1.dat

```
n = 5;
w = [1, 5, 2 , 3, 4];
c= [5, 8, 4,  7, 9];
k = 8;
```

Knapsack2.dat

```
n = 6;
w = [1, 5, 2 , 3, 4, 5];
c= [5, 8, 4,  7, 9, 5];
k = 8;
```

OPL IDE



Example:

- Two projects

  First example 2010

  knapsack_carlo

- *knapsack_carlo* contains
  - one model (*knapsack.mod*)
  - two instances (*knapsack1.dat, knapsack2.dat*)
  - two configurations (*configuration1, configuration2)*

- Configuration1 contains model *knapsack.mod* and instance *knapsack1.dat*

- Configuration2 contains model *knapsack.mod* and instance *knapsack2.dat*

s-t path problem

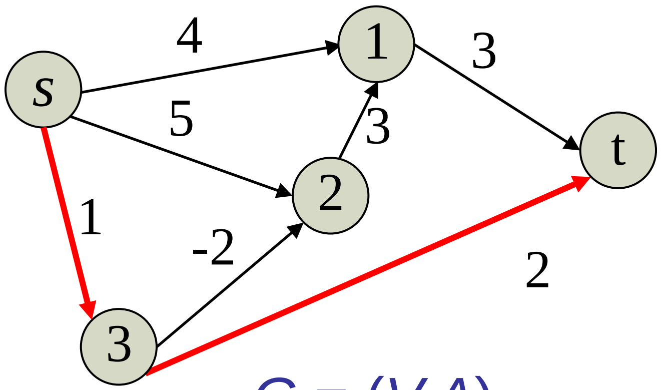
$$\min \sum_{uv \in A} c_{uv} x_{uv}$$

$$\sum_{us \in \delta^-_D(s)} x_{us} - \sum_{su \in \delta^+_D(s)} x_{su} = -1 \qquad s$$

$$\sum_{ut \in \delta^-_D(t)} x_{ut} - \sum_{tu \in \delta^+_D(t)} x_{tu} = 1 \qquad t$$

$$\sum_{uv \in \delta^-_D(v)} x_{uv} - \sum_{vu \in \delta^+_D(v)} x_{vu} = 0 \qquad v \in V-\{s,t\}$$

$$x \geq 0 \quad x \in R^A$$



$G = (V,A)$

$\min c^T x$

$Ax = b$

$x \geq 0$

$$\begin{array}{c} \\ s \\ 1 \\ 2 \\ 3 \\ t \end{array} \begin{array}{c} \begin{array}{ccccccc} s1 & s2 & s3 & 1t & 21 & 32 & 3t \end{array} \\ \begin{pmatrix} -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array} \quad c = \begin{pmatrix} 4 \\ 5 \\ 1 \\ 3 \\ 3 \\ -2 \\ 2 \end{pmatrix} \quad b = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

s-t path problem

$$\min c^T x$$

$$Ax = b$$

$$x \geq 0$$

$$\begin{array}{c} \\ s \\ 1 \\ 2 \\ 3 \\ t \end{array} \begin{array}{c} \begin{array}{ccccccc} s1 & s2 & s3 & 1t & 21 & 32 & 3t \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 \end{pmatrix} \end{array} \qquad c = \begin{pmatrix} 4 \\ 5 \\ 1 \\ 3 \\ 3 \\ -2 \\ 2 \end{pmatrix} \qquad b = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

```
int n = ...; // number of nodes
int m = ...; // number of edges
range nodes = 1..n;
range edges = 1..m;
int A[nodes][edges] = ...;
int b[nodes] = ...;
int c[edges] = ...;
dvar float+ x[edges];

minimize sum(j in edges) c[j]*x[j];
subject to {
  forall (i in nodes)
     y: sum(j in edges) A[i][j]*x[j] == b[i];
}
```

```
n = 5;
m = 7;
b = [-1, 0, 0, 0, 1];
c = [4, 5, -1, 3, 3, -2, 32];
A = [ [-1,-1,-1, 0, 0, 0, 0]
        [1,  0, 0,-1, 1, 0, 0]
        [0,  1, 0, 0,-1, 1, 0]
        [0,  0, 1, 0, 0,-1,-1]
        [0,  0, 0, 1, 0, 0, 1]];
```

Aggegating data: Tuples

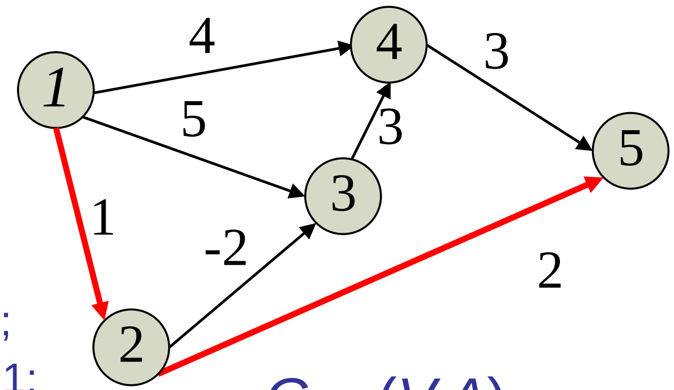
- Several related data can be clustered together in *tuples*

```
tuple Point {
   float x;
   float y;
}

Point P1 = <1,2>;
int a = P1.x;
Int b = P1.y;
{Point} points = { <1,2>, <2,3>, <4.1, 5.2>};
```

Exploiting sparsity: tuples

$$\textbf{max } y_t - y_s$$
$$y_v - y_u \le c_{uv} \quad \text{for all } uv \in A$$

```
int Nvert = ...;
range Verts = 1..Nvert;
int source = …;
int sink = …;
tuple arc {
  int u;
  int v;
  float w;
}
{arc} Arcs = ...;

dvar float y[Verts];
maximize
  y[sink] - y[source];

subject to {

  forall(e in Arcs) y[e.v] - y[e.u] <= e.w;
}
```



G = (V,A)

```
Nvert = 5;
source = 1;
sink = 5;

Arcs = {
    <1 2 1>,
    <1 3 5>,
    <1 4 4>,
    <2 3 -2>,
    <2 5 2>,
    <3 4 3>,
    <4 5 3>,
    };
```

Expression

- Conditional expressions can be used in loops and in summations.

```
int Nvert = ...;
range Verts = 1..Nvert;
int source = ...;
int sink = ...;
tuple arc {
  int u;
  int v;
  float w;
}
{arc} A = ...;
 dvar float+ x[A];
```

$$\min \sum_{uv \in A} c_{uv} x_{uv}$$

$$\sum_{us \in \delta_D^-(s)} x_{us} - \sum_{su \in \delta_D^+(s)} x_{su} = -1 \quad s$$

$$\sum_{ut \in \delta_D^-(t)} x_{ut} - \sum_{tu \in \delta_D^+(t)} x_{tu} = 1 \quad t$$

$$\sum_{uv \in \delta_D^-(v)} x_{uv} - \sum_{vu \in \delta_D^+(v)} x_{vu} = 0$$

$$x \geq 0 \quad x \in R^A$$

```
 minimize sum (e in A) e.w * x[e];
subject to {
  forall (z in Verts : z != source && z!= sink)
      z: sum (e in A: e.v == z) x[e] - sum (e in A: e.u == z) x[e] == 0 ;
   s: sum (e in A: e.v == source) x[e]  - sum (e in A: e.u == source) x[e] == -1;
   t: sum (e in A: e.v == sink) x[e]  - sum (e in A: e.u == sink) x[e] == 1;
}
```