An introduction to CPLEX
And its C++ API

Who? Alberto Santini[1]

From? [1]Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione, Università di Bologna

When? September 29, 2014

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Summary

What is CPLEX?

Modelling in CPLEX

A typical CPLEX workflow

Additional resources

Summary

What is CPLEX?

CPLEX is a **solver** for various types of optimisation problems:

- Convex programming
  - Linear programming
  - Second-order cone programming
- Integer and mixed-integer programming
- Quadratic programming
- Constraint programming

What is CPLEX?

CPLEX is a **solver** for various types of optimisation problems:

- Convex programming
  - Linear programming
  - Second-order cone programming
- Integer and mixed-integer programming
- Quadratic programming
- Constraint programming

In red are the words you **should** know!

What is CPLEX?

A **solver** is a programme that takes as its input:

And gives as its output:

What is CPLEX?

A **solver** is a programme that takes as its input:

- A structured version of a mathematical model

And gives as its output:

What is CPLEX?

A **solver** is a programme that takes as its input:

- A structured version of a mathematical model

And gives as its output:

- The optimal solution to the model — If we are lucky!

What is CPLEX?

A **solver** is a programme that takes as its input:

- A structured version of a mathematical model

And gives as its output:

- The optimal solution to the model — If we are lucky!
- Or, a feasible solution to the model, a measure of its quality and some "bounds" on the optimal solution value — That's more reasonable...

What is CPLEX?

A **solver** is a programme that takes as its input:

- A structured version of a mathematical model

And gives as its output:

- The optimal solution to the model — If we are lucky!
- Or, a feasible solution to the model, a measure of its quality and some "bounds" on the optimal solution value — That's more reasonable...
- Or, at least some information on how a solution might look like? — Please!

What is CPLEX?

The most general MIP we will solve with CPLEX:

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} c_i x_i & & \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_{ij} x_i \leq b_i & & \forall j = 1, \ldots, m \\
& l_i \leq x_i \leq u_i & & \forall i = 1, \ldots, n \\
& x_i \in D_i & & \forall i = 1, \ldots, n
\end{aligned}
$$

What is CPLEX?

The most general MIP we will solve with CPLEX:

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} c_i x_i & & \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_{ij} x_i \leq b_i & & \forall j = 1, \ldots, m \\
& l_i \leq x_i \leq u_i & & \forall i = 1, \ldots, n \\
& x_i \in D_i & & \forall i = 1, \ldots, n
\end{aligned}
$$

- There are n variables and m constraints

What is CPLEX?

The most general MIP we will solve with CPLEX:

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} c_i x_i & & \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_{ij} x_i \leq b_i & & \forall j = 1, \ldots, m \\
& l_i \leq x_i \leq u_i & & \forall i = 1, \ldots, n \\
& x_i \in D_i & & \forall i = 1, \ldots, n
\end{aligned}
$$

- There are n variables and m constraints
- l_i and u_i are lower and upper bounds on x_i

What is CPLEX?

The most general MIP we will solve with CPLEX:

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} c_i x_i & & \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_{ij} x_i \leq b_i & & \forall j = 1, \ldots, m \\
& l_i \leq x_i \leq u_i & & \forall i = 1, \ldots, n \\
& x_i \in D_i & & \forall i = 1, \ldots, n
\end{aligned}
$$

- There are n variables and m constraints
- l_i and u_i are lower and upper bounds on x_i
- D_i is the domain of definition of the variable x_i:
  - $D_i = \mathbb{R}$ — continuous variable
  - $D_i = \mathbb{Z}$ — integer variable
  - $D_i = \{0, 1\}$ — binary variable

What is CPLEX?

Notice that since arrays in C++ start with the index 0, we will "shift back" the indices in the previous model by 1 position:

$$\min \sum_{i=0}^{n-1} c_i x_i$$

$$\text{s.t.} \sum_{i=0}^{n-1} a_{ij} x_i \leq b_i \qquad \forall j = 0, \ldots, m-1$$

$$l_i \leq x_i \leq u_i \qquad \forall i = 0, \ldots, n-1$$

$$x_i \in D_i \qquad \forall i = 0, \ldots, n-1$$

Summary

Modelling in CPLEX

We need to translate the mathematical model into computer instructions.

CPLEX supports two ways of doing this:

- Modelling **by rows**
- Modelling **by columns**

Modelling by rows

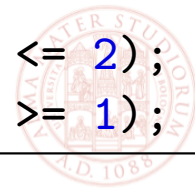
Modelling **by rows** is the most natural way, but only works well for small instances:

$$
\begin{array}{lrrr}
\min & 1x_0 & +1x_1 & -1x_2 \\
\text{s.t.} & 4x_0 & +2x_1 & -5x_2 \leq 2 \\
 & -1x_0 & +1x_1 & +1x_2 \geq 1
\end{array}
$$

Becomes:

```
// Obj function
model.add(IloMinimize(x[0] + x[1] - x[2]));

// 2 constraints (rows)
cst.add(4 * x[0]  + 2 * x[1]  - 5 * x[2] <= 2);
cst.add(-   x[0]  +     x[1]  +     x[2] >= 1);
```



Modelling by columns

Modelling **by columns** is a bit trickier, but it scales well to big problems:

$$\begin{array}{lrrr} \min & 1x_0 & +1x_1 & -1x_2 \\ \text{s.t.} & 4x_0 & +2x_1 & -5x_2 \leq 2 \\ & -1x_0 & +1x_1 & +1x_2 \geq 1 \end{array}$$

Becomes:

```
// Obj function
IloObjective obj = IloMinimize(env);

// 2 constraints (rows)
cst.add(IloRange(env, -IloInfinity, 2.0));
cst.add(IloRange(env, 1.0, IloInfinity));

// 3 variables (columns)
x.add(IloNumVar(obj( 1.0) + cst[0]( 4.0) + cst[1](-1.0)));
x.add(IloNumVar(obj( 1.0) + cst[0]( 2.0) + cst[1]( 1.0)));
x.add(IloNumVar(obj(-1.0) + cst[0](-5.0) + cst[1]( 1.0)));
```

Modelling by columns

Modelling **by columns** is a bit trickier, but it scales well to big problems:

$$\begin{array}{lrrr} \min & 1x_0 & +1x_1 & -1x_2 \\ \text{s.t.} & 4x_0 & +2x_1 & -5x_2 \leq 2 \\ & -1x_0 & +1x_1 & +1x_2 \geq 1 \end{array}$$

Becomes:

```
// Obj function
IloObjective obj = IloMinimize(env);

// 2 constraints (rows)
cst.add(IloRange(env, -IloInfinity, 2.0));
cst.add(IloRange(env, 1.0, IloInfinity));

// 3 variables (columns)
x.add(IloNumVar(obj( 1.0) + cst[0]( 4.0) + cst[1](-1.0)));
x.add(IloNumVar(obj( 1.0) + cst[0]( 2.0) + cst[1]( 1.0)));
x.add(IloNumVar(obj(-1.0) + cst[0](-5.0) + cst[1]( 1.0)));
```

Modelling by columns

Modelling **by columns** is a bit trickier, but it scales well to big problems:

$$
\begin{array}{lrrr}
\min & 1x_0 & +1x_1 & -1x_2 \\
\text{s.t.} & 4x_0 & +2x_1 & -5x_2 \leq 2 \\
 & -1x_0 & +1x_1 & +1x_2 \geq 1
\end{array}
$$

Becomes:

```
// Obj function
IloObjective obj = IloMinimize(env);

// 2 constraints (rows)
cst.add(IloRange(env, -IloInfinity, 2.0));
cst.add(IloRange(env, 1.0, IloInfinity));

// 3 variables (columns)
x.add(IloNumVar(obj( 1.0) + cst[0]( 4.0) + cst[1](-1.0)));
x.add(IloNumVar(obj( 1.0) + cst[0]( 2.0) + cst[1]( 1.0)));
x.add(IloNumVar(obj(-1.0) + cst[0](-5.0) + cst[1]( 1.0)));
```

Modelling by columns

Modelling **by columns** is a bit trickier, but it scales well to big problems:

$$
\begin{array}{lrrr}
\min & 1x_0 & +1x_1 & -1x_2 \\
\text{s.t.} & 4x_0 & +2x_1 & -5x_2 \leq 2 \\
 & -1x_0 & +1x_1 & +1x_2 \geq 1
\end{array}
$$

Becomes:

```
// Obj function
IloObjective obj = IloMinimize(env);

// 2 constraints (rows)
cst.add(IloRange(env, -IloInfinity, 2.0));
cst.add(IloRange(env, 1.0, IloInfinity));

// 3 variables (columns)
x.add(IloNumVar(obj( 1.0) + cst[0]( 4.0) + cst[1](-1.0)));
x.add(IloNumVar(obj( 1.0) + cst[0]( 2.0) + cst[1]( 1.0)));
x.add(IloNumVar(obj(-1.0) + cst[0](-5.0) + cst[1]( 1.0)));
```

Modelling by columns

Modelling **by columns** is a bit trickier, but it scales well to big problems:

$$
\begin{array}{lrrrl}
\min & 1x_0 & +1x_1 & -1x_2 & \\
\text{s.t.} & 4x_0 & +2x_1 & -5x_2 & \leq 2 \\
 & -1x_0 & +1x_1 & +1x_2 & \geq 1
\end{array}
$$

Becomes:

```
// Obj function
IloObjective obj = IloMinimize(env);

// 2 constraints (rows)
cst.add(IloRange(env, -IloInfinity, 2.0));
cst.add(IloRange(env, 1.0, IloInfinity));

// 3 variables (columns)
x.add(IloNumVar(obj( 1.0) + cst[0]( 4.0) + cst[1](-1.0)));
x.add(IloNumVar(obj( 1.0) + cst[0]( 2.0) + cst[1]( 1.0)));
x.add(IloNumVar(obj(-1.0) + cst[0](-5.0) + cst[1]( 1.0)));
```

Summary

A typical CPLEX workflow

A typical programme that uses CPLEX contains the following parts:

- Data input and preprocessing
- Creation of the CPLEX environment and model objects
- Creation of rows and columns
- Solve
- Analysis of the results
- Destruction of the CPLEX environment object

Data input and preprocessing

For complex models, data are usually stored in data files or databases.

More often than not, it's in a format that needs preprocessing before being used to build a model.

Creation of the CPLEX environment and model objects

The very first thing to do is to setup a CPLEX environment by creating an object of class `IloEnv`:

```
IloEnv env;
```

The environment provides optimised memory management and is going to be used for everything CPLEX-related. You can think of an object of class `IloEnv` as a pointer.

Creation of the CPLEX environment and model objects

The next thing is to create the backbone for our model, an object of class `IloModel`:

```
IloModel model(env);
```

We will add to the model variables (`IloNumVar`), constraints (`IloRange`) and the objective function (`IloObjective`) through the method `add`. For example:

```
model.add(variable);
model.add(objective_function);
model.add(constraint);
```

Creation of rows and columns

But before adding these elements to the model, we need to create them, by creating objects of the following classes:

```
IloNumVar this_is_a_variable;
  // For example x
IloNumVarArray this_is_an_array_of_variables;
  // For example x[0] ... x[100]
IloRange this_is_a_constraint;
  // For example x[1] + x[2] <= 2
IloRangeArray this_is_an_array_of_constraints;
  // For example x[i] + x[i+1] <= 2 for all i = 0 ... 99
IloObjective this_is_the_objective_function;
  // For example x[0] + ... + x[100]
```

Creation of rows and columns

A complete example, when modelling by rows:

```
IloEnv env;
IloModel model(env);

IloNumVarArray x(env);

for(int i = 0; i < 3; i++) {
  std::stringstream name;
  name << "x_" << i;

  x.add(IloNumVar(
    env, 0, 1, IloNumVar::Bool, name.str().c_str()
  ));
}
model.add(IloMinimize(env, x[0] + x[1] - x[2]));

IloRangeArray constraints(env);

constraints.add( x[0] + 2 * x[1] - x[2] <= 2);
constraints.add(-x[0] +     x[1] + x[2] >= 1);

model.add(constraints);
```

Creation of rows and columns

Notice that in the previous example:

- We didn't need to explicitly add the variables to the model, as they are added through the objective function and the constraints
- We declared **boolean** variables, therefore their LB was 0 and their UB was 1
- We gave a name to our variables: `x_0`, `x_1`, `x_2`
- We didn't give a name to our constraints (but it's very much possible to do so)

Creation of rows and columns

Names for variables and constraints are particularly useful if we decide to export our model to a human-readable file, via:

```
IloCplex cplex(model);
cplex.exportModel("model.lp");
```

An `.lp` file is a good way (for small models) to check that we didn't make any mistake in our code, especially when we model by columns.

Creation of rows and columns

The .lp file for the model we just created looks like this:

```
Minimize
 obj: x_0 + x_1 - x_2
Subject To
 c1: x_0 + 2 x_1 - x_2 <= 2
 c2: - x_0 + x_1 + x_2 >= 1
Bounds
 0 <= x_0 <= 1
 0 <= x_1 <= 1
 0 <= x_2 <= 1
Binaries
 x_0  x_1  x_2
End
```

Solve

In order to start the solver, we call the `solve()` method of an `IloCplex` object:

```
IloCplex cplex(model);
if(cplex.solve()) {
  // Everything OK
} else {
  // Some problem occurred
}
```

CPLEX will print many informative messages and updates us on the status of the solution process.

Analysis of the results

In both cases, wether `solve()` was successful or not, we can inquire about the status of the solver:

```
using std::cout; using std::cerr; using std::endl;
if(cplex.solve()) {
  cout << "Cplex completed!" << endl;
  cout << "Status: " << cplex.getStatus() << endl;
  cout << "Obj value: " << cplex.getObjValue() << endl;
} else {
  cerr << "Cplex error!" << endl;
  cerr << "Status: " << cplex.getStatus() << endl;
  cerr << "Cpx status: " << cplex.getCplexStatus() << endl;
}
```

Analysis of the results

After the solver completes, there is a variety of information we can retrieve. The most important ones are:

- Is the problem feasible?
- Was the optimal solution found?
- If not, what are the UB and LB?
- What are the values of the variables in the optimal solution?
- If we are solving an LP, what are the values of the dual variables?
- How many branch-and-bound nodes were explored?

Destruction of the CPLEX environment object

At the very end, we must remember to free up all the memory used by CPLEX (and it's usually a lot), with a call to:

```
env.end();
```

Summary

Additional resources

You can find CPLEX, course material, configuration help, example programs and references at:

`http://137.204.57.204/`

Remember that CPLEX is provided through the IBM Academic Initiative and therefore you can only use it for academic non-commercial purposes.