ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INSTITUTO DE INVESTIGACIÓN TECNOLÓGICA

Application Development

Andrés Ramos

Universidad Pontificia Comillas, Madrid
http://www.iit.upcomillas.es/aramos/
Andres.Ramos@upcomillas.es

Contents

Last decade in optimization

- “In the last decade, new advances in algorithms have been as important as the impressive advances in computer technology” George L. Nemhauser (1994)

- “The technology improvements in algorithms, modeling languages, software, and hardware have made the methodology accessible, easy to use, and fast. So the Age of Optimization has arrived” George L. Nemhauser (1994)

Algorithmic advances in LP with CPLEX

- Since CPLEX 3.0 in 1994 to CPLEX 7.0 in 2000 a specific problem has improved 28 times, using the dual simplex method
- Between version CPLEX 1.0 of 1988 and 7.0 of 2000 there has been a global improvement, in software and algorithmic, of 10000 times
- As a reference the improvement in hardware performance has been of the same order of magnitude
- LP problems with up to 190.000 constraints, 630.000 variables y 1.900.000 non zero elements “easily” solved in a PC with 256 MB of RAM memory

Today it can be solved in seconds problems that would have required years ten years ago

Stages in developing an optimization model

1. Mathematical formulation
   - Type of problem to be solved (LP, MIP, NLP, DP, SP, MCP)
   - Estimated size of the problem (rows x columns: 1000x1000, 10000x10000, 100000x100000)

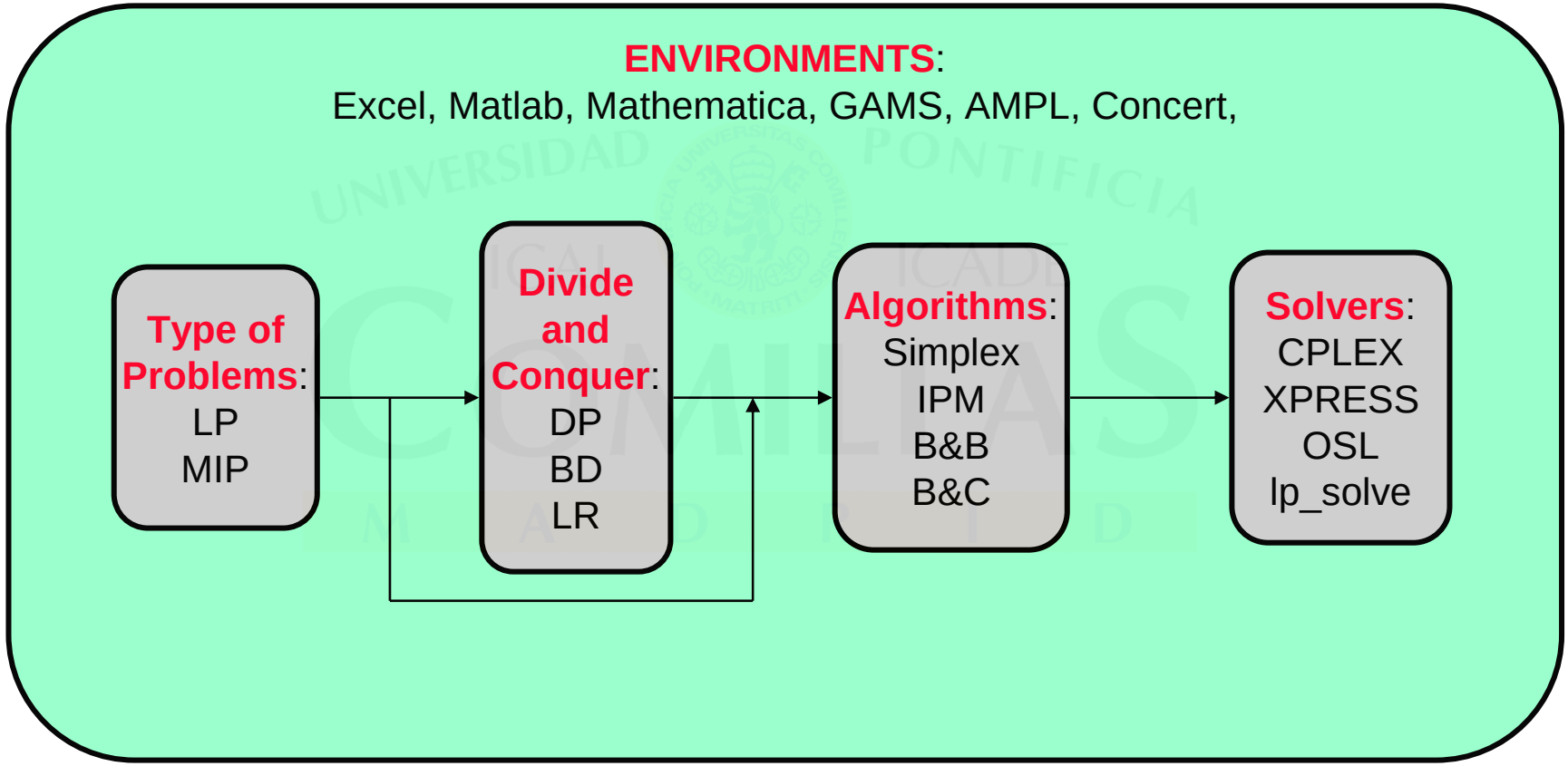
2. Model development
   - Several alternatives or environments are possible
   - Public domain or commercial codes
   - Robustness of the method and solver

3. Analysis and evaluation of the results, sensitivity analysis

Alternatives for developing optimization models

- General purpose programming language
  - C (CPLEX from ILOG, OSL from IBM)
  - C++ (Concert from ILOG, LINDO API from LINDO Systems, OptiMax 2000 from Maximal Software, FLOPC++ from Universidade de Aveiro)
  - Java, Visual Basic, FORTRAN 90
- Numeric and symbolic environments or languages
  - Excel, Matlab, Mathematica
- Algebraic modeling languages
  - GAMS, AMPL, AIMMS, XPRESS-MP, MPL
- In *OR/MS Today* (www.orms-today.com) once a year there are surveys about the different optimization environments available and their characteristics

Solution process of an optimization problem



Solvers in spreadsheets

- Advantages
  - Easy to use
  - Total integration with the spreadsheet
  - Familiarity with the environment that eases the explanation of the model and the results
  - Easy presentation of graphical results
- Disadvantages
  - Do not induce good programming practice
  - Present difficulties for verification, update and model documentation
  - Do not allow to model complex or huge problems

Optimization library in C++

- Advantages
  - Solution time is critical
  - Separation of problem and solution algorithm
  - Facilitate the creation and manipulation of optimization problems
  - Allow the use of specific optimization algorithms
  - Allow the model implementation in special software or hardware
- Disadvantages
  - Increased developing difficulty and resource (man-hours) consumption for model development and maintenance

Algebraic modeling languages. Advantages (i)

- High level languages for compact formulation of big and complex models
- Ease prototype development
- Improve modelers' productivity
- Structure good modeling habits
- Split data and mathematical structure of the model
- Formulation independent of the problem size
- Model independent of solvers

Algebraic modeling languages. Advantages (ii)

- Ease continuous reformulation
- Documentation embedded and simultaneous with modeling
- Allow implementation of advanced algorithms
- Easy implementation of NLP, MIP or MCP problems
- Portability among platforms and operating systems (Windows, Linux, Sun Solaris, HP UX, DEC Digital Unix, IBM AIX, SGI IRIX)

Algebraic modeling languages. Disadvantages

- Not suitable for sporadic uses with small size problems
- Not suitable for solving directly huge problems (1.000.000 x 1.000.000)

Future tendencies

- Visual interface for formulation
- Strong interface with spreadsheets and data bases

- Constraint logic programming extensions
- Direct resolution of stochastic optimization problems
  - MSLiP, OSLSE, DECIS, SLP-IOR
- Automatic selection of the most suitable optimization method and solver

Contents

DC optimal power flow without losses. Hypothesis

- Unit commitment decisions known
- Thermal units have linear variable costs
- Hydro plant output divided in two blocks:
  - Scheduled output with no variable cost
  - Emergency output with an opportunity cost (i.e., water value)
- Only active power is considered
- Line flow proportional to the angular difference between voltage angles of extreme nodes
- Losses are neglected

DC optimal power flow without losses (i)

- **Minimize operation variable costs in one hour:**
  - Variable costs of thermal units
  - Opportunity costs (water value) of hydro plants when producing above the scheduled output
  - Variable cost of non served energy

$$\sum_{t=1}^{T} v_t GTR_t + \sum_{h=1}^{H} v_h GHE_h + \sum_{n=1}^{N} v_n PNS_n$$

  - Data: v_t: generation variable cost of thermal unit t.

    v_h: opportunity cost of hydro emergency output h.

    v_n: variable cost of non served energy.
  - Variables: GTR_t output of thermal unit t.

    GHE_h emergency output of hydro plant h.

    PNS_n non served power in node n.

DC optimal power flow without losses (ii)

- **Simple variable bounds of the generation subsystem:**
  - Minimum and maximum output of thermal unit *t*:

$$\underline{GTR_t} \le GTR_t \le \overline{GTR_t}$$

  - Maximum scheduled output of hydro plant *h*:

$$0 \le GHP_h \le \overline{GHP_h}$$

  - Maximum emergency output of hydro plant *h*:

$$0 \le GHE_h \le \left(\overline{GHM_h} - \overline{GHP_h}\right)$$

  - Non served energy lower than node demand *n*.

$$0 \le PNS_n \le D_n$$

DC optimal power flow without losses (iii)

- **Electric network model:**
  - **1st Kirchoff's law:** Balance between generation and demand in each node:

$$\sum_{t\in n} GTR_t + \sum_{h\in n}(GHP_h + GHE_h) + PNS_n + \sum_{i=1}^{I} F_{i\to n} - \sum_{j=1}^{J} F_{n\to j} = D_n \quad \forall n$$

  - **2nd Kirchoff's law:** Active power flow through each line:

$$\frac{X_{i\to j}}{S_B} F_{i\to j} = \theta_i - \theta_j \quad \forall i \to j$$

  - **Simple bounds for flows**: $-\overline{F_{i\to j}} \le F_{i\to j} \le \overline{F_{i\to j}}$

DC optimal power flow without losses (iv)

- **Electric network model, alternative formulation:**
  - **1st Kirchoff's law:**

$$\sum_{t\in n} GTR_t + \sum_{h\in n}(GHP_h + GHE_h) + PNS_n +$$

$$\sum_{i=1}^{I}(\theta_i - \theta_n)S_B / X_{i\to n} - \sum_{j=1}^{J}(\theta_n - \theta_j)S_B / X_{n\to j} = D_n \quad \forall n$$

  - **Angular difference bounds as constraints**:

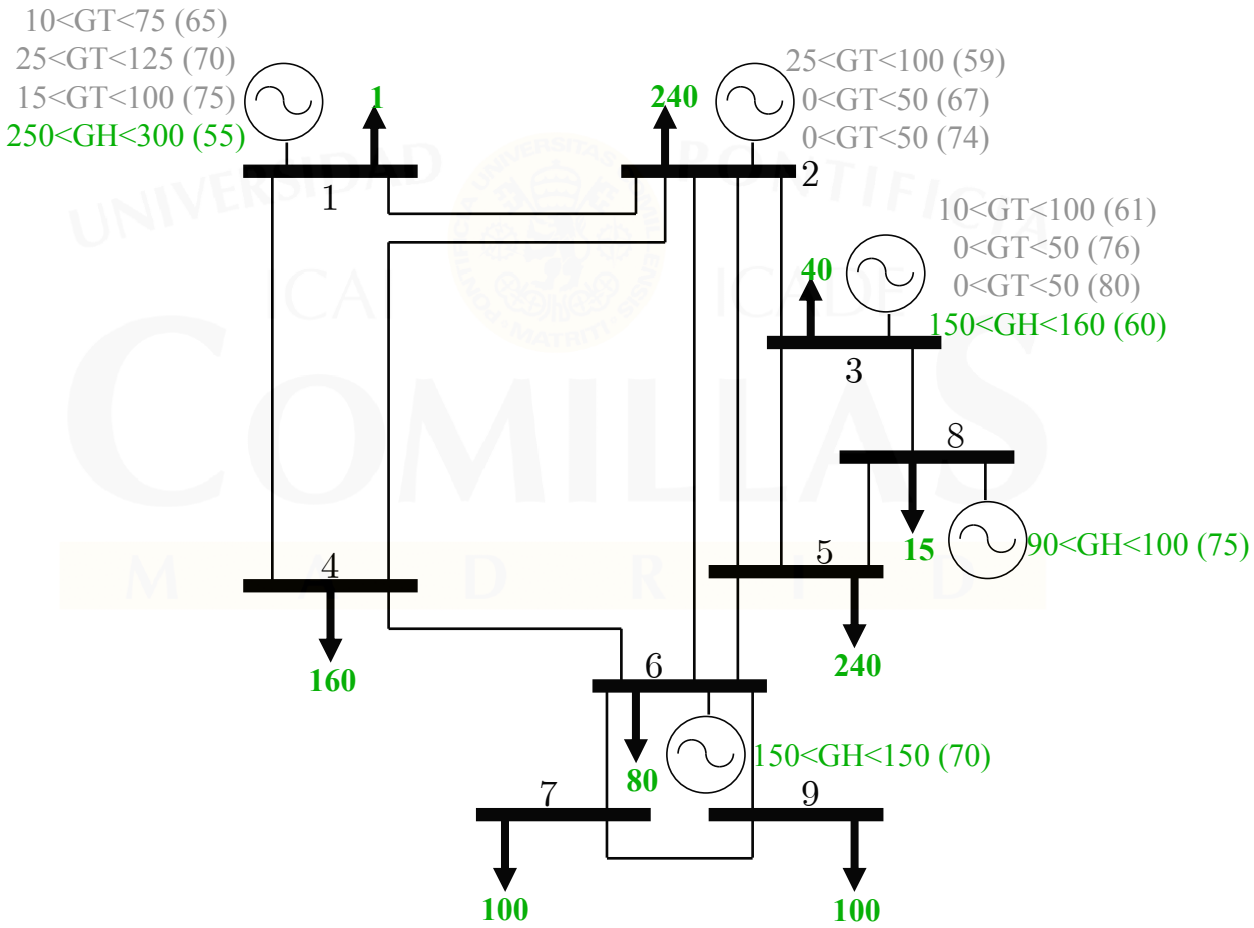
$$\theta_i - \theta_j \le \overline{F_{i\to j}} \frac{X_{i\to j}}{S_B} \quad \forall i \to j$$

$$\theta_i - \theta_j \ge -\overline{F_{i\to j}} \frac{X_{i\to j}}{S_B} \quad \forall i \to j$$

- This formulation has less variables (no flows), but more constraints (maximum angular difference).

Case example

9 thermal units, 4 hydro units, 9 nodes, 14 lines



Case example

Formulation 1	
24 Constraints	**50 Variables**
1 objective function	1 objective variable
9 balance equations (1st Kirchoff's law)	9 thermal unit output
14 flow equations (2nd Kirchoff's law)	4 hydro unit scheduled and 4 emergency output
	9 non served energy
	14 line flows
	9 voltage angles

Formulation 2	
38 Constraints	**36 Variables**
1 objective function	1 objective variable
9 balance equations (1st Kirchoff's law)	9 thermal unit output
28 maximum angular difference	4 hydro unit scheduled and 4 emergency output
	9 non served energy
	9 voltage angles

Case example in three flavors

1. Spreadsheet:
   - EXCEL from Microsoft and solver from Frontline Systems.
2. Algebraic modeling language:
   - GAMS from GAMS Development Corp.
3. Optimization classes:
   - Concert from ILOG

Contents

Basic characteristics. Data and variable bounds

- In the worksheet it is represented the objective function, the constraint matrix, the bounds on the variables and their optimal values

			Thermal unit data								
			GTR1	GTR2	GTR3	GTR4	GTR5	GTR6	GTR7	GTR8	GTR9
Variable cost / Opportunity cost (euro/MWh)			65	70	75	59	67	74	61	76	80
Minimum output (MW)			10	25	14	25			10		
Maximum / Scheduled output (MW)			75	125	100	100	50	50	100	50	50
Maximum output of hydro unit (MW)											

	Hydro unit data				Hydro unit data			Non served energy data								
GHP1	GHP2	GHP3	GHP4	GHE1	GHE2	GHE3	GHE4	PNS1	PNS2	PNS3	PNS4	PNS5	PNS6	PNS7	PNS8	PNS9
				55	60	70	75	1500	1500	1500	1500	1500	1500	1500	1500	1500
250	140	140	90	50	20		10	1	240	40	160	240	80	100	15	100
				300	160	140	100									

Electric line data													
NODE1-NODE2	NODE1-NODE4	NODE2-NODE3	NODE2-NODE4	NODE2-NODE5	NODE2-NODE6	NODE3-NODE5	NODE3-NODE8	NODE4-NODE6	NODE5-NODE6	NODE5-NODE8	NODE6-NODE7	NODE6-NODE9	NODE7-NODE9
-500	-500	-500	-500	-500	-500	-500	-500	-500	-500	-500	-500	-500	-500
500	500	500	500	500	500	500	500	500	500	500	500	500	500

Constraint matrix

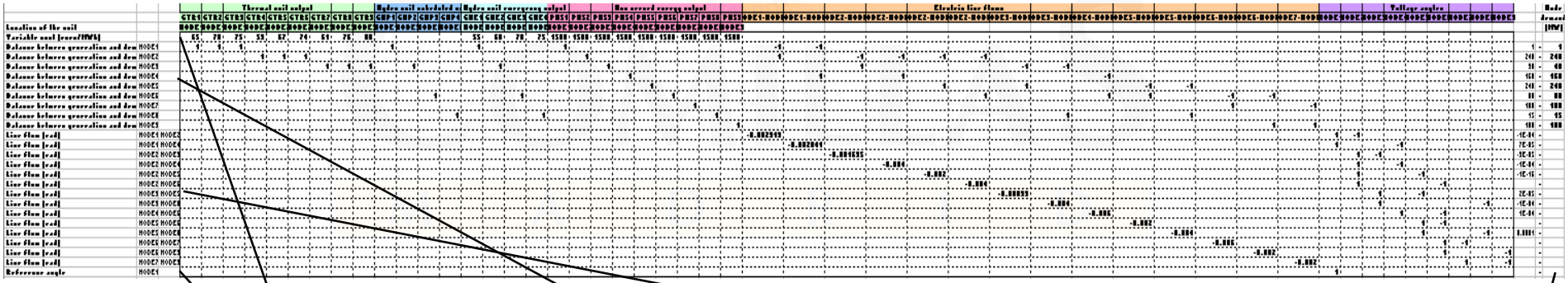
Thermal unit variables

Hydro unit variables

Line flow variables

Non served energy variables

Voltage angle variables



Objective function

1st Kirchoff's law

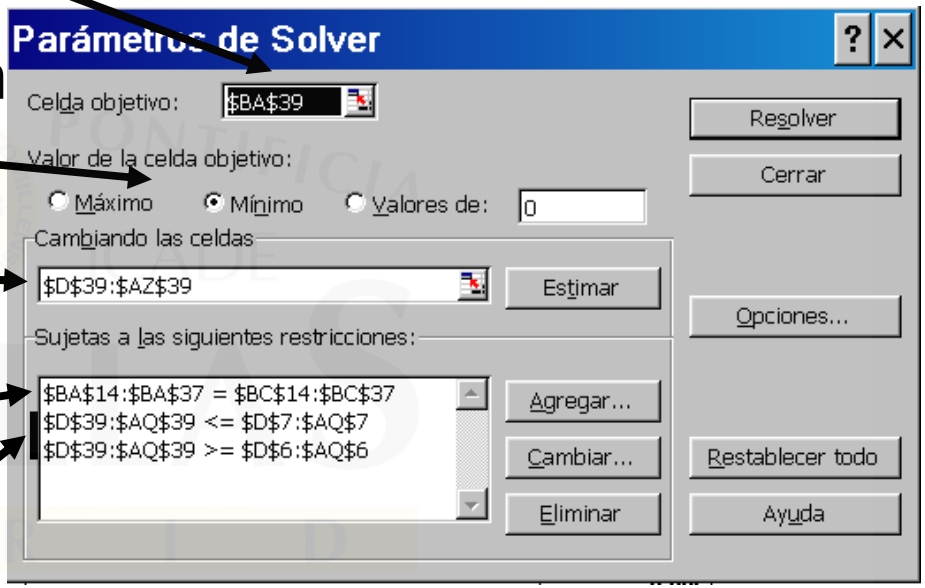
2nd Kirchoff's law

Reference angle

Constraint bounds

Solver parameters

- Objective variable cell
- Maximization or minimization
- Variables cells
- Constraint cells
- Variable bound cells



Solution output

- Optimal solution of variables:
  - thermal generation
  - hydro generation
  - non served energy
  - line flows
  - voltage angles
- Objective variable value

Contents

GAMS

- `sets`, `alias`
- Data: `scalars`, `parameters`, `table`
- Variables: `positive`, `free`, `integer`, `binary`
- `equations`
- `model`
- `solve`

UNIVERSIDAD PONTIFICIA ICAI ICADE COMILLAS MADRID

Contents

Development in Concert

- Development stages of the optimization model and solution by the solver:
  1. Environment creation
  2. Data input to the model
  3. Model building
  4. Model extraction for an algorithm
  5. Solution
  6. Solution output

ILOG Concert summary

- Basic structure
  - Environment `IloEnv`, myname`.end`
  - Model `IloModel`
- Basic functions
  - Data types `IloNumArray`
  - Variable type `IloNumVar`, myname`.setLB`, myname`.setUB`, myname`.setName`, myname`.getLB`, myname`.getUB`, myname`.getName`, myname`.getValue`
  - Expressions `IloExpr`
  - Objective function `IloObjective`, `IloMinimize`
  - Constraints `IloRange`, mymodel.`add`
  - Solver `IloCplex`, `cplex.extract`, `cplex.solve`, `cplex.getValue`, `cplex.getObjValue`, `cplex.exportModel`, `cplex.getObjValue`, , `cplex.getDualValue`
  - File management MyFile`.open`, MyFile`.is_open`, MyFile`.close`, myname`.getSize`

Environment creation

- Environment: class where all the objects representing the modeling objects of an optimization model (data, variables, equations, objective function, solver) are built
- Creation

```
IloEnv MyName
```

- Deletion

```
MyName.end
```

- Example

```
IloEnv MyEnvironment
```

Data representation

- Allows the creation of structure arrays of any type. It is useful for building data tables.

`IloNumArray`	number vector
`IloNumVar`	variable vector
`IloRangeArray`	constraint vector

- Example

```
typedef IloArray<IloNumArray> NumMatrix;
NumMatrix MyData(MyEnvironment,2);
MyData[0] = IloNumArray(MyEnvironment,3,1.0,2.4,4.5);
MyData[1] = IloNumArray(MyEnvironment,3,7.0,4.0,3.0);
```

Data input from files to arrays

- Natural data input for applications

```
NumMatrix DATNUD(MyEnvironment);
  {ifstream MyFile;
  MyFile.open("C:\\node_data.dat");
  MyFile >> DATNUD;
  MyFile.close();}
```

- Example

```
[[  1, 1500]
 [240, 1500]
 [ 40, 1500]
 [160, 1500]
 [240, 1500]
 [ 80, 1500]
 [100, 1500]
 [ 15, 1500]
 [100, 1500]]
```

Variable declaration

- When building a model, the variables belong to the working environment. Afterwards, the equations are added to the model
- Concert allows multiple ways to build a variable array

```
IloNumVarArray MyName (environment,size,lower_bound,
  upper_bound,variable_type)
```

- Example

```
IloNumVarArray
  GTR(MyEnvironment,gr,0,IloInfinity,ILOFLOAT);
```

Model creation and constraint addition (i)

```
IloModel DCOPF(MyEnvironment)
```

- Addition of constraints my be done directly or by creating a constraint that is added to the model afterwards. It is not necessary to create a constraint type object to be added to the model. However, this practice is recommended for later modification of the associated parameters to the constraints
- Example

```
DCOPF.add(x[0]+x[1]+x[2] <= 3);
```

Model creation and constraint addition (ii)

- A constraint type object is mainly composed by an expression (expression type object) and a lower and upper bounds
- Example

```
IloExpr v(MyEnvironment);
v += x[1] + x[2] + x[3];
IloRange MyConstraint(MyEnvironment,-IloInfinity,v,3);
DCOPF.add(MyConstraint);
```

Objective creation

- Analogously as adding constraints to a model, the objective creation may be done directly or by an objective type object added to the model afterwards
- Example
  - Case 1

```
IloExpr obj(MyEnvironment);
obj += COSTE;
DCOPF.add(IloMinimize(MyEnvironment,obj))
```

  - Case 2

```
IloExpr v(MyEnvironment);
IloObjective
  obj(MyEnvironment,v,IloMinimize);
DCOPF.add(obj)
```

Solver creation and model solution

- Concert offers the possibility of solving the same model with different solvers with different types of algorithms. They are encapsulated under the class `IloCplex`, that contains the algorithms primal simplex, dual simplex, interior point, and branch and bound.
- Example

```
IloCplex MyAlgorithm(MyEnvironment)
MyAlgorithm.extract(DCOPF)
MyAlgorithm.solve()
```

- The model extraction creates the constraint matrix that otherwise it would be created by the user. This is the main advantage of the existence of models

Selection of solution algorithm

- Done by using the member function `setRootAlgorithm` defined over the optimization algorithm class `IloCplex`
- Example

```
MyAlgorithm.setRootAlgorithm(IloCplex::Primal);
MyAlgorithm.setRootAlgorithm(IloCplex::Dual);
MyAlgorithm.setRootAlgorithm(IloCplex::Barrier);
```

- For MIP problems the algorithm used is branch and bound, with Gomory cuts, rounding, GUB and clique.
- All these cuts can be activated or deactivated by specific control parameters

Data output

- The main function is `getValue`, defined over the optimization algorithm class `IloCplex`.
- The variable object does not represent a position where the current value is stored, but this value is associated to the algorithm
- Getting the objective function value

```
MyAlgorithm.getObjectiveValue()
```

- Getting a variable value

```
MyAlgorithm.getValue(variable_name)
```

- Getting a dual variable value of a constraint

```
MyAlgorithm.getDualValue(constraint_name)
```

Data output

- Allows to show the results by screen or by writing to disk. The sentences present the same structure that those of data input from files to data arrays
- Example of screen output

```
env.out() << “Value of the objective function” <<
  MyAlgorithm.getObjValue() << endl;
```

- Example of file output

```
{ofstream MyFile;
   MyFile.open("salida.DAT");
   MyFile << “Value of the objective function” <<
             MyAlgorithm.getObjValue() << endl;
   MyFile.close();}
```

To infinite and beyond