UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INSTITUTO DE INVESTIGACIÓN TECNOLÓGICA

2ª Jornada sobre Modelos de Planificación y Operación de los Sistemas de Energía Eléctrica

Introducción a Concert Technology

Andrés Ramos
Santiago Cerisola
Jesús María Latorre

Índice

1. Introducción
2. Modelado con tecnología ILOG Concert.
   Caso ejemplo

Introducción (I)

Opciones al resolver problemas de optimización:

- Lenguaje de modelado algebraico (GAMS, AMPL, ...)
- Librerías de funciones o clases
- Optimizador interactivo
- Código propio

Ventajas:

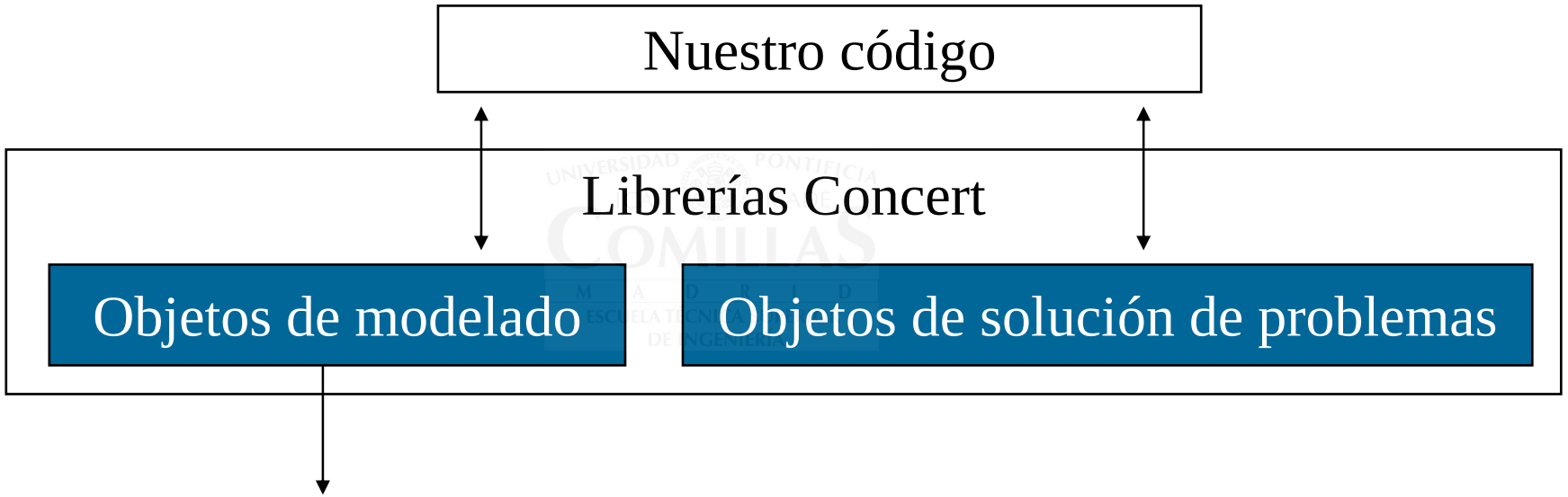
- Es lo más rápido
- Puede ser muy cercano al modelado
- Pueden usarse diferentes algoritmos
- Algoritmos complejos

Inconvenientes:

- Lenguajes de programación generales
- Menos flexible

Introducción (II)

Estructura de una aplicación que use Concert Technology:



`IloNumVar`	Variables numéricas
`IloRange`	Restricciones
`IloObjective`	Función objetivo
`IloModel`	Modelo

Fases de una aplicación

1. Creación de un entorno
2. Obtención de los datos del modelo
3. Construcción del modelo
4. Extracción del modelo para un algoritmo
5. Resolución
6. Obtención de los valores de la solución
7. Finalización

Caso ejemplo

Problema de programación entera mixta:

$$\min -0.3x - 1.5y - z$$

$$x + y \leq 3.7$$

$$y + z \leq 5.2$$

$$0 \leq x \leq 5$$

$$y \geq 0,\ y \in \mathbb{Z}$$

$$z \geq 0,\ z \in \mathbb{Z}$$

1. Creación de un entorno

- El entorno sirve para:
  - Manejo de la memoria
  - Identificación de los objetos

- Todos los objetos deben construirse asociados a un entorno.

- Creación:

```
IloEnv MiEntorno();
```

- Borrado:

```
MiEntorno.end();
```

2. Datos del modelo

- Representación de los datos en matrices:

`IloNumArray`	Vector de valores numéricos reales
`IloNumVarArray`	Vector de variables continuas
`IloIntArray`	Vector de variables enteras
`IloRangeArray`	Vector de restricciones

- Creación de vectores propios:

```
typedef IloArray<NombreClase> NuevoTipo;
```

Usos:

  - Creación de vectores de tipos propios.
  - Creación de vectores multidimensionales:

```
typedef IloArray<IloNumArray> NumMatrix;
```

2. Datos del modelo

- Lectura de datos de archivo: con el operador >>

- Ejemplo de archivo de entrada: `[datos.txt]`

```
[[  0.225 , 0.153 , 0.162 ],
 [  0.225 , 0.162 , 0.126 ]]
```

- Ejemplo de código de lectura:

```
ifstream MiArchivo;
IloNumArray Datos(MiEntorno);
MiArchivo.open("datos.txt");
MiArchivo >> Datos;
MiArchivo.close();
```

3. Creación del modelo

- Se necesitan las variables que forman parte del modelo:

```
IloNumVar X(MiEntorno, 0.0, 5.0, ILOFLOAT);
IloNumVar Y(MiEntorno, 0.0, IloInfinity, ILOINT);
IloNumVar Z(MiEntorno, 0.0, IloInfinity, ILOINT);
```

- Se debe crear una variable de tipo `IloModel` que agrupe todos los objetos que representan el problema:

```
IloModel MiModelo(MiEntorno);
```

- Al modelo se añaden las restricciones:
  - Directamente
  - Creando objetos intermedios

3. Creación del modelo

- Creación de la función objetivo:

$$\min -0.3x - 1.5y - z$$

- La clase `IloExpr` recoge expresiones del modelo.
- Las funciones `IloMinimize` e `IloMaximize` forman la función objetivo a partir de una expresión.

- Código:

```
IloExpr ExObj(MiEntorno);
ExObj = -0.3 * X - 1.5 * Y - 1.0 * Z;
MiModelo.add(IloMinimize(MiEntorno, ExObj));
ExObj.end();
```

3. Creación del modelo

- La clase `IloRange` representa una restricción. Se crea a partir de una expresión y sus límites.
- Creación de las restricciones:

$$x + y \leq 3.7$$

```
IloExpr Ex1(MiEntorno);
Ex1 = 1.0 * X + 1.0 * Y;
IloRange Eq1(MiEntorno, -IloInfinity, Ex1, 3.7);
MiModelo.add(Eq1); Ex1.end();
```

$$y + z \leq 5.2$$

```
IloExpr Ex2(MiEntorno);
Ex2 = 1.0 * Y + 1.0 * Z;
IloRange Eq2(MiEntorno, -IloInfinity, Ex2, 5.2);
MiModelo.add(Eq2); Ex2.end();
```

4 y 5. Extracción y resolución del modelo

- La clase `IloCplex` resuelve los problemas y controla el algoritmo de resolución.
- Primero hay que extraer el modelo:

```
IloCplex MiCplex(MiEntorno);
MiCplex.extract(MiModelo);
```

- Para resolver el problema:

```
MiCplex.solve();
```

- Se puede elegir el algoritmo de resolución del problema con `IloCplex::setRootAlgorithm`, p.ej.:

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

6. Acceso al resultado

- La lectura de la solución del problema se hace mediante las funciones miembro de la clase `IloCplex`.
- Se puede saber el estado del algoritmo con `IloCplex::getStatus()`
- Para saber el valor de la solución del problema `IloCplex::getObjValue()`
- Para leer el valor de una variable del problema en la solución `IloCplex::getValue(`*Variable*`)`
- Código:

```
cout << "Objetivo: " << MiCplex.getObjValue() << endl;
cout << " X: " << MiCplex.getValue(X) << endl;
cout << " Y: " << MiCplex.getValue(Y) << endl;
cout << " Z: " << MiCplex.getValue(Z) << endl;
```

7. Finalización

Al terminar es necesario liberar la memoria de los objetos:

- Al terminar el objeto de la clase `IloEnv`, se libera la memoria de los objetos extraíbles (variables, restricciones, ...)
- Terminar los vectores (`IloArray::end()`) libera la memoria del vector pero no de sus elementos.
- Las expresiones (`IloExpr`) deben ser liberadas en cuanto no se necesiten.

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INSTITUTO DE INVESTIGACIÓN TECNOLÓGICA

2ª Jornada sobre Modelos de Planificación y Operación de los Sistemas de Energía Eléctrica

Introducción a Concert Technology

Andrés Ramos
Santiago Cerisola
Jesús María Latorre