
# Sesión 6
# Desarrollo de modelos de optimización con ILOG CPLEX

**Andrés Ramos**

**Santiago Cerisola**

**Jesús María Latorre**

# Índice

# Avances algorítmicos en LP con CPLEX

- Desde CPLEX 3.0 en 1994 a CPLEX 7.0 en 2000 la mejora en el método simplex dual para un problema concreto ha sido de 28 veces
- Entre la versión de CPLEX 1.0 de 1988 y la 7.0 del 2000 se observa una mejora global, de software y algorítmica, de 10000 veces
- Como referencia, la mejora en rendimiento del hardware ha sido del mismo orden de magnitud

Hoy se pueden resolver en segundos lo que hace una docena de años se habría resuelto en años

# Uso del optimizador ILOG CPLEX

- Optimizador muy rápido y fiable para la resolución de problemas LP, MIP, QP
- Versión actual 8.0

- Uso interactivo
- Uso a través de una llamada
  - Lenguaje algebraico de modelado (e.g., GAMS)
  - Llamada a funciones desde C++ (ILOG Concert Technology)
  - Llamada a funciones desde C (CPLEX Callable Library)
  - Llamada a funciones desde Java (en el futuro)

# Lenguaje algebraico de modelado GAMS (i)

- Ventajas
  - Lenguajes de alto nivel para formulación compacta de modelos grandes y complejos
  - Facilitan desarrollo de prototipos
  - Mejorar productividad de modeladores
  - Estructuran buenos hábitos de modelado
  - Separan datos de estructura matemática de modelo
  - Formulación independiente del tamaño
  - Modelo independiente de optimizadores
  - Facilitan reformulación continua
  - Documentación simultánea al modelo
  - Permiten implantación de algoritmos avanzados
  - Implantación fácil de problemas NLP, MIP, MCP
  - Portabilidad entre plataformas y sistemas operativos

# Lenguaje algebraico de modelado GAMS (ii)

- Inconvenientes
  - No son adecuados para usos esporádicos con problemas de pequeño tamaño
  - No son adecuados para resolución directa problemas de tamaño gigantesco (1.000.000 x 1.000.000)

# Modelado con tecnología ILOG Concert C++

- Ventajas
  - Tiempo de solución es crítico
  - Separa problema y algoritmo de solución
  - Facilita la creación y manipulación de problemas de optimización
  - Permiten el uso de algoritmos de optimización específicos
  - Posibilidad de implantación del modelo en un entorno software o hardware especial

- Inconvenientes
  - Mayor dificultad y consumo de recursos para el mantenimiento del modelo

# Flujo de cargas óptimo en DC sin pérdidas (i)

- **Minimizar los costes variables de operación** en una hora:
  - costes variables de los grupos térmicos
  - costes de oportunidad de los grupos hidráulicos cuando producen por encima de su potencia programada.
  - coste variable de la potencia no suministrada.

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

  - Datos: $v_t$: coste variable de generación del grupo térmico $t$.

    $v_h$: coste de oportunidad de la hidráulica de emergencia.

    $v_h$: coste variable de la potencia no suministrada.
  - Variables: $GTR_t$ potencia producida por el grupo térmico $t$.

    $GHE_h$ potencia hidráulica de emergencia del grupo $h$.

    $PNS_n$ potencia no suministrada en el nudo $n$.

# Flujo de cargas óptimo en DC sin pérdidas (ii)

- **Cotas de las variables** del equipo generador:
  - Potencia térmica máxima y mínima del grupo *t*:

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

  - La potencia hidráulica programada máxima del grupo *h*:

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

  - La potencia hidráulica de emergencia máxima del grupo *h*:

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

  - La potencia no suministrada como mucho será la demanda del nudo *n*.

$$0 \le PNS_n \le D_n$$

# Flujo de cargas óptimo en DC sin pérdidas (iii)

- Modelado de la red de transporte:
  - **1ª Ley Kirchoff**: Balance entre generación y demanda de nudo:

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

  - **2ª Ley Kirchoff**: Flujo de potencia activa por las líneas:

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

  - **Cotas de los flujos**: $-\overline{F_{i\to j}} \le F_{i\to j} \le \overline{F_{i\to j}}$

# Flujo de cargas óptimo en DC sin pérdidas (iv)

- Modelado de la red de transporte:
  - **Formulación alternativa de la 1ª Ley Kirchoff**:

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

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

  - **Límites térmicos de las líneas** como restricciones:

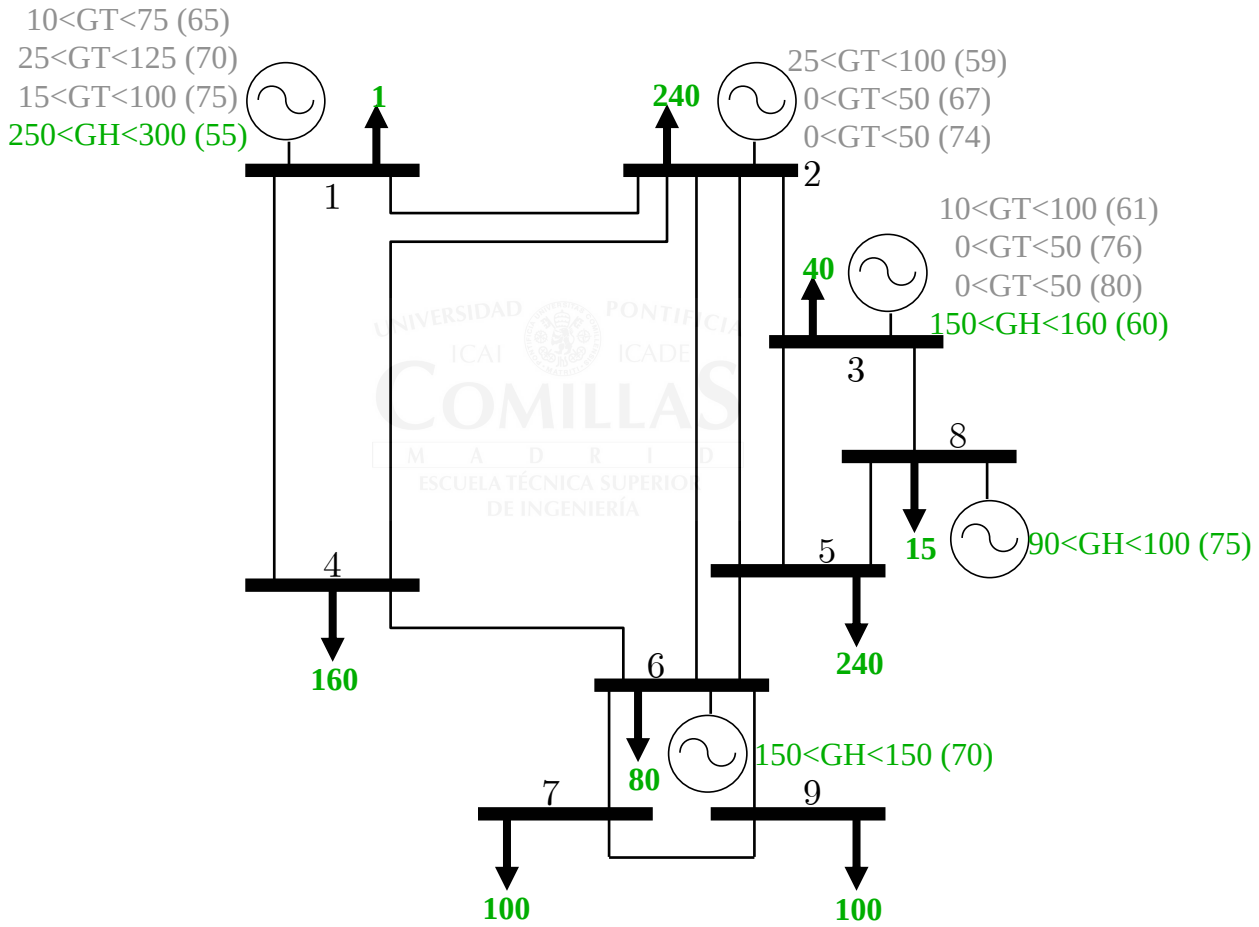
$$\theta_i - \theta_j \leq \overline{F_{i \to j}} \frac{X_{i \to j}}{S_B}$$

$$\theta_i - \theta_j \geq -\overline{F_{i \to j}} \frac{X_{i \to j}}{S_B}$$

- Esta formulación tiene menos variables, pero más restricciones.

# Caso de estudio

9 generadores térmicos, 4 hidráulicos, 9 nudos, 14 líneas



# Case example

| Formulación 1 | |
|---|---|
| **24 Restricciones** | **50 Variables** |
| 1 función objetivo | 1 variable objetivo |
| 9 ecuaciones de balance (1ª ley de Kirchoff) | 9 producciones generadores térmicos |
| 14 ecuaciones de flujo (2ª ley de Kirchoff) | 4 hidráulica programada y 4 hidráulica de emergencia |
| | 9 potencia no suministrada |
| | 14 flujos por las líneas |
| | 9 ángulos de tensión |

| Formulación 2 | |
|---|---|
| **38 Restricciones** | **36 Variables** |
| 1 función objetivo | 1 variable objetivo |
| 9 ecuaciones de balance (1ª ley de Kirchoff) | 9 producciones generadores térmicos |
| 28 diferencias angulares máximas | 4 hidráulica programada y 4 hidráulica de emergencia |
| | 9 potencia no suministrada |
| | 9 ángulos de tensión |

# Presentación de los códigos para caso de estudio

- Comentario y presentación de código completo escrito en GAMS
- Comentario y presentación de código completo escrito en Concert
- Código escrito en C

# Índice

# GAMS

- Conjuntos e índices `sets, alias`
- Datos `scalars, parameters, table`
- Variables `positive, free, integer, binary variable`
- Ecuaciones `equations`
- Modelo `model`
- Optimizador `solve`

# Índice

# Desarrollo con Concert

- Las fases del desarrollo de un modelo de programación matemática con Concert y optimización del mismo utilizando un optimizador son resumidas en los siguientes puntos
  - Creación de un entorno
  - Introducción de datos al modelo
  - Construcción de un modelo
  - Extracción del modelo para un algoritmo
  - Resolución
  - Obtención de valores de la solución

# Creación del entorno

- Entorno: clase sobre la que se construyen los objetos que representan objetos de modelado de un modelo de programación matemática (datos, variables, ecuaciones, función objetivo, optimizador).
- Creación

  `IloEnv minombre`
- Destrucción

  `minombre.end`
- Ejemplo particular de aplicación

  `IloEnv mientorno`

# Representación de los datos

- Permite la creación de arrays de estructuras de cualquier tipo. En particular esto es útil para la construcción de tablas de datos.

| | |
|---|---|
| `IloNumArray` | vector de números |
| `IloNumVar` | vector de variables |
| `IloRangeArray` | vector de restricciones |

- Ejemplo particular

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

# Lectura de datos desde ficheros a arrays

- Introducción natural de los datos para aplicaciones

```
NumMatrix DATNUD(mientorno);
  {ifstream mifichero;
  mifichero.open("C:\\datos_de_nudos.dat");
  mifichero >> DATNUD;
  mifichero.close();}
```

- Ejemplo de fichero de datos

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

# Declaración de variables del entorno

- En la construcción de un modelo, las variables pertenecen al entorno en el que se está trabajando. Posteriormente son las ecuaciones las que se añadirán al modelo
- Concert presenta multitud de constructores para un array de variables

```
IloNumVarArray minombre (mientorno,tamaño,cota
  inferior,cota superior,tipo de variable)
```

- Ejemplo

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

# Creación del modelo y adición de restricciones (I)

```
IloModel FC(mientorno)
```

- La adición de restricciones puede hacerse de modo directo o vía la creación de una restricción que posteriormente se añada al modelo. No es necesario la creación de un objeto de tipo restricción que se añada al modelo. Sin embargo, esta práctica es recomendable para la posterior modificación de parámetros asociados a las restricciones
- Ejemplo

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

# Creación del modelo y adición de restricciones (II)

- Un objeto de tipo restricción está formado principalmente por una expresión (objeto de tipo expresión) y una cota superior y otra inferior
- Ejemplo

```
IloExpr v(mientorno);
v += x[1] + x[2] + x[3];
IloRange mirestriccion(mientorno,-IloInfinity,v,3);
FC.add(mirestriccion);
```

# Creación del objetivo

- De modo análogo a la adición de restricciones a un modelo, la creación de un objetivo puede hacerse de modo directo o mediante la creación de un objeto de tipo objetivo posteriormente añadido al modelo
- Ejemplo
  - Situación 1

```
IloExpr obj(mientorno);
obj += COSTE;
FC.add(IloMinimize(mientorno, obj))
```

  - Situación 2

```
IloExpr v(mientorno);
IloObjective obj(mientorno,v,IloMinimize);
FC.add(obj)
```

# Creación del optimizador y resolución del modelo

- Concert ofrece la posibilidad de resolver el mismo modelo con diferentes optimizadores con diferentes algoritmos. Estos están encapsulados bajo la clase `IloCplex`, que contiene los algoritmos simplex primal, simplex dual, punto interior, branch and bound.
- Ejemplo

```
IloCplex mialgoritmo(mientorno)
mialgoritmo.extract(FC)
mialgoritmo.solve()
```

- La extracción del modelo crea la matriz de restricciones que sin el uso de Concert debería ser introducida por el usuario. Ésta es la diferencia y ventaja principal de Concert, la existencia de modelos

# Modificación del algoritmo de resolución

- Se lleva a cabo utilizando la función miembro `setRootAlgorithm` definida sobre la clase de algoritmos de optimización `IloCplex`
- Ejemplo

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

- Si el problema presenta variables enteras el algoritmo utilizado es el branch and bound, con aplicación por omisión de planos de corte Gomory, rounding, GUB y clique.
- La activación o no de estos cortes se puede hacer mediante el uso de funciones específicas.

# Obtención de resultados

- La función principal para ello es `getValue`, definida sobre la clase de algoritmos de optimización `IloCplex`.
- Debe destacarse que el objeto variable no presenta un campo donde se almacena el valor temporal que presenta, sino que ese valor está asociado al algoritmo.
- Obtención del valor de la función objetivo

```
mialgoritmo.getObjectiveValue()
```

- Obtención del valor de una variable

```
mialgoritmo.getValue(nombre de la variable)
```

- Obtención del valor dual de una restricción

```
mialgoritmo.getDualValue(nombre de la restricción)
```

# Presentación de resultados

- Posibilidad de presentación de los resultados por pantalla o escritura en disco. Las instrucciones presentan la misma estructura que la lectura de datos desde ficheros a arrays de datos
- Ejemplo de salida por pantalla

```
env.out() << “Valor de la función objetivo” <<
  mialgoritmo.getObjValue() << endl;
```

- Ejemplo de salida a fichero

```
{ofstream mifichero;
  mifichero.open("salida.DAT");
  mifichero << “Valor de la función objetivo” <<
            mialgoritmo.getObjValue() << endl;
  mifichero.close();}
```

# ILOG Concert

- Estructura básica
  - mientorno `IloEnv`, nombre`.end`
  - Modelo `IloModel`
- Funciones básicas
  - Tipos de datos `IloNumArray`
  - Tipos de variables `IloNumVar`, nombre`.setLB`, nombre`.setUB`, nombre`.setName`, nombre`.getLB`, nombre`.getUB`, nombre`.getName`, nombre`.getValue`
  - Expresiones `IloExpr`
  - Función objetivo `IloObjective`, `IloMinimize`
  - Restricciones `IloRange`, nombre_modelo.`add`
  - Optimizador `IloCplex`, `cplex.extract`, `cplex.solve`, `cplex.getValue`, `cplex.getObjValue`, `cplex.exportModel`, `cplex.getObjValue`, , `cplex.getDualValue`
  - Manejo ficheros fichero`.open`, fichero`.is_open`, fichero`.close`, nombre`.getSize`

# Índice

# ILOG CPLEX

- Estructura básica
  - mientorno `CPXENVptr`
  - Problema `CPXLPptr`
- Funciones básicas
  - Apertura del optimizador `CPXopenCPLEX`
  - Creación de un problema `CPXcreateprob`
  - Añade variables y función objetivo `CPXnewcols`
  - Añade restricciones `CPXaddrows`
  - Escritura de un problema `CPXwriteprob`
  - Resolución de un problema `CPXlpopt`
  - Escritura de resultados `CPXwritesol`
  - Liberación del problema `CPXfreeprob`
  - Liberación del optimizador `CPXcloseCPLEX`

# Índice

# Parámetros de control de ILOG CPLEX

- Permiten un control más detallado de las acciones del optimizador
  - Generales de control y de escritura
  - Para problemas LP
  - Para problemas MIP

# Parámetros generales de control

| | |
|---|---|
| `lpmethod` | Selección del método de optimización (simplex primal y dual, barrera) |
| `preind` | Realización de un preproceso de eliminación de variables y restricciones |
| `scaind` | Control del escalado del problema |
| `epopt` | Tolerancia en el criterio de optimalidad |
| `eprhs` | Tolerancia en el criterio de factibilidad |
| `quality` | Presenta propiedades numéricas de la solución del problema |
| `tilim` | Tiempo máximo de ejecución |
| `aggind` | Realiza pasadas de preproceso |

# Parámetros generales de control

| | |
|---|---|
| `workdir` | Directorio de almacenamiento temporal del árbol |
| `workmem` | Cantidad máxima de memoria a utilizar |
| `barooc` | Método de punto interior con escritura a disco |
| `nodefileind` | Especifica dónde guardar los resultados de la exploración del árbol |
| `precompress` | Comprime el problema original |

# Parámetros generales de escritura

| | |
|---|---|
| `simdisplay` | Presenta información de la resolución de un problema LP por el método simplex |
| `bardisplay` | Presenta información de la resolución de un problema LP por el método de punto interior |
| `mipdisplay` | Presenta información de la resolución de un problema MIP |
| `writemps` | Escribe el problema en formato MPS |

# Programación lineal

$$\min_x c^T x$$

$$Ax = b$$

$$x \geq 0$$

$$x \in \mathbb{R}^n, c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$$

- Método simplex primal (se mueve de vértice en vértice de la región factible hacia la solución óptima manteniendo la factibilidad) y dual (se mueve disminuyendo la infactibilidad manteniendo la condición de optimalidad)
- Método de punto interior (primal-dual predictivo correctivo) (se mueve por el interior de la región factible hasta alcanzar el vértice óptimo)

# Algoritmos de programación lineal

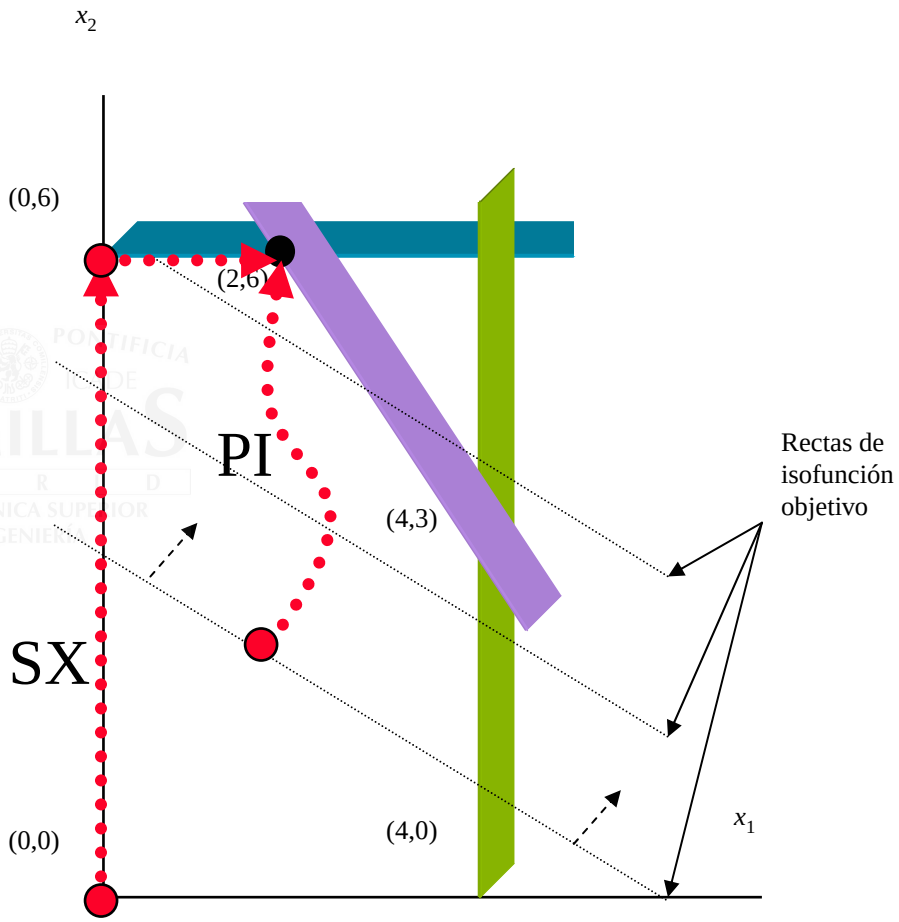
$$\max \quad z = 3x_1 + 5x_2$$

$$x_1 \leq 4$$

$$2x_2 \leq 12$$

$$3x_1 + 2x_2 \leq 18$$

$$x_1, x_2 \geq 0$$



# Selección del algoritmo

- No existe una regla clara
- No hay regla para determinar qué algoritmo simplex es más eficiente. Muy sensible a la estructura del problema
- Probar y observar

| | |
|---|---|
| Método simplex | Hasta 10.000 x 10.000 |
| Método simplex | Análisis de sensibilidad, problemas MIP |
| Método de punto interior | Desde 10.000 x 10.000<br>hasta 100.000 x 100.000 |
| Métodos de descomposición | Más de 100.000 x 100.000 |

# Parámetros de problemas LP

| | |
|---|---|
| `iis` | Detección del conjunto mínimo de infactibilidades |
| `baralg` | Selección del método de punto interior |
| `barcrossalg` | Selecciona el método de cruce para encontrar solución básica factible óptima |
| | |
| `barcolnz` | Número de elementos no nulos en una columna densa de la matriz de restricciones |
| `epmrk` | Tolerancia de Markowitz. Mejora las propiedades numéricas |

# Programación lineal entera mixta

$$\max \; Z = 4x_1 - 2x_2 + 7x_3 - x_4$$

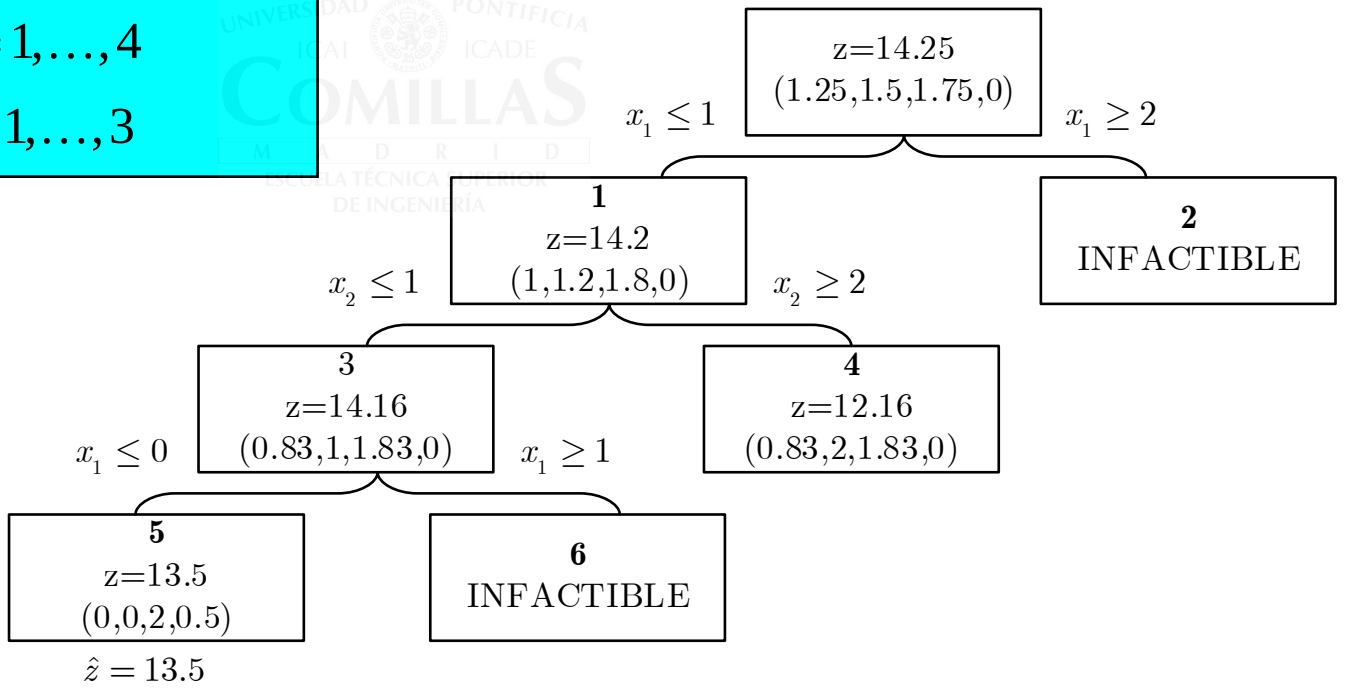
$$\begin{aligned} x_1 \quad & \quad & +5x_3 \quad & \quad & \leq 10 \\ x_1 \quad & +x_2 \quad & -x_3 \quad & \quad & \leq 1 \\ 6x_1 \quad & -5x_2 \quad & \quad & \quad & \leq 0 \\ -x_1 \quad & \quad & +2x_3 \quad & -2x_4 \quad & \leq 3 \end{aligned}$$

$$x_j \geq 0 \qquad j = 1,\ldots,4$$

$$x_j \text{ enteras} \qquad j = 1,\ldots,3$$

$$Z^* = 13.5$$

$$(x_1, x_2, x_3, x_4) = (0, 0, 2, 0.5)$$



# Parámetros para problemas MIP

| | |
|---|---|
| `mipemphasis` | Criterio de exploración: buscar factibilidad o buscar optimalidad |
| `nodesel` | Criterio de selección del nodo a explorar |
| `varsel` | Criterio de selección de la variable a ramificar en cada nodo |
| `mipstart` | Utilización de una solución entera inicial |
| `startalg` | Algoritmo utilizado en la resolución del problema LP inicial |
| `subalg` | Algoritmo utilizado para resolver los problemas LP en cada nodo |
| `epint` | Tolerancia de integralidad |
| `epgap` | Criterio de optimalidad relativa entre solución entera y mejor solución LP posible |
| `cuts` | Utiliza o no diferentes tipos de cortes |