



# Instructivo de CPLEX

Modelos y Optimización I

Redactado por Pablo Romano

Facultad de Ingeniería de la Universidad de Buenos Aires

Versión 1.0 - Mayo 2012

# Índice

1. Introducción.....	3
1.1. Download.....	3
1.2. Licencia académica .....	3
2. Problema.....	4
2.1. Enunciado .....	4
2.2. Modelo .....	5
3. Uso de CPLEX .....	6
3.1. Creando un nuevo proyecto.....	6
3.2. Resolución básica utilizando CPLEX.....	7
3.3. Ejecución y análisis de una corrida.....	8
4. Separación de los datos del modelo.....	11
5. Separación de datos del modelo en CPLEX.....	13
5.1. Modelo .....	13
5.2. Sección de datos .....	15
5.3. Ejecución.....	15
6. Funciones.....	16
7. Limitando el tiempo de ejecución .....	17

## **1. Introducción**

El presente documento pretende ser introductorio al uso del programa IBM ILOG CPLEX que es utilizado para la resolución de problemas de programación lineal.

### **1.1. Download**

Entrar a:

<http://www.ibm.com/developerworks/university/membership/join.html>

Seguir los pasos de Step 1 “Register for a universal IBM ID and password”

Una vez registrado, ir a la opción del menú:

Support & downloads → Download → Trials & demos, hacer click en “IBM ILOG CPLEX” o ir directamente a

<http://www14.software.ibm.com/webapp/download/search.jsp?pn=IBM+LOG+CPLEX>

Bajar la última versión de CPLEX para el sistema operativo donde se lo va a instalar.

### **1.2. Licencia académica**

Solicitar al docente una licencia académica.

## 2. Problema

Para ilustrar el uso de CPLEX, se utilizará como ejemplo el ejercicio tipo 5.2 cuyo enunciado se incluye a continuación.

### 2.1. Enunciado

En una fábrica de medias se desea analizar la operación de un sector integrado por tres equipos E1, E2, E3 donde se procesan los productos A, B, C. Los tiempos de proceso de los productos son los del siguiente cuadro, medidos en horas de equipo por docena de producto.

	A	B	C
Equipo 1	0,8	0,8	0,3
Equipo 2	0,6	1,2	0
Equipo 3	0,6	1,0	0,6

Se ha determinado además, la disponibilidad mensual de cada uno de los equipos. Esta importa respectivamente 160, 180 y 110 horas. Asimismo, se estima en 100 docenas mensuales la cantidad demandada máxima del producto A, y en 120 docenas mensuales la cantidad demandada máxima del producto B.

Por otra parte, la Dirección de la empresa desea producir como mínimo 80 docenas mensuales del producto B. El margen de beneficio de cada producto es de 50\$ por docena de A, 40\$ por docena de B y 30\$/docena de C. El programa óptimo es el que hace máximo el margen total de beneficio.

## 2.2. Modelo

### Procesamiento

$$0,8A + 0,8B + 0,3C < 160 \quad (\text{procEq1})$$

$$0,6A + 1,2B < 180 \quad (\text{procEq2})$$

$$0,6A + 1B + 0,6C < 110 \quad (\text{procEq3})$$

### Demandas máximas y mínimas

$$A < 100 \quad (\text{demMaxA})$$

$$B < 120 \quad (\text{demMaxB})$$

$$B > 80 \quad (\text{demMinB})$$

### Definición del funcional

$$\text{MAX} \rightarrow Z = 50\$A + 40\$B + 30\$C$$

*Figura 1: Modelo*

### 3. Uso de CPLEX

#### 3.1. Creando un nuevo proyecto

Para crear un nuevo proyecto ir a File → New → OPL Project

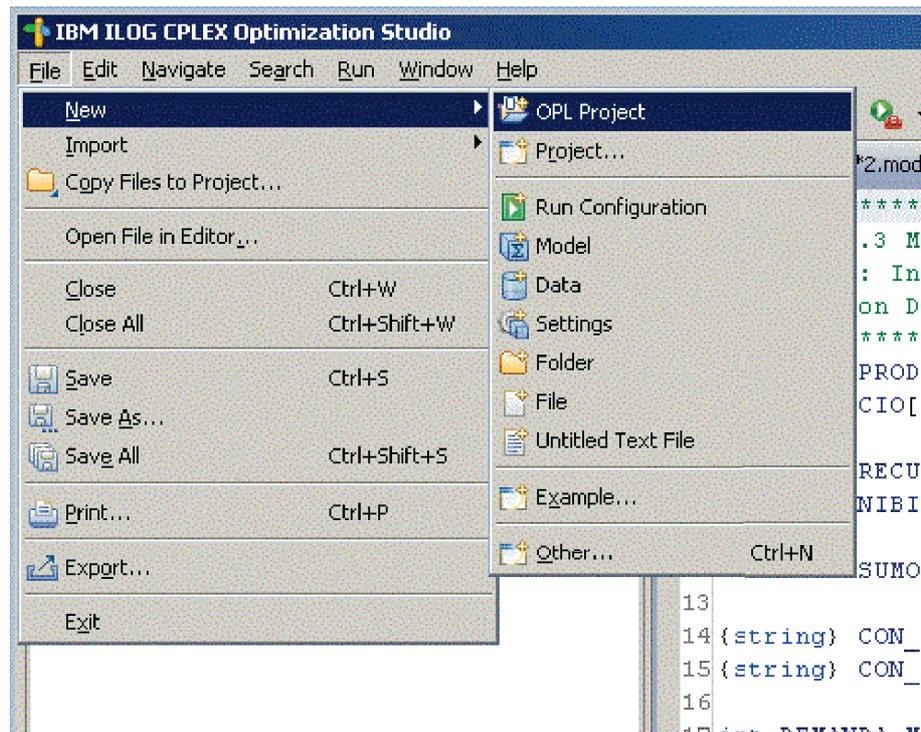


Figura 2: Nuevo proyecto

Indicar el nombre del proyecto y marcar todas las opciones.

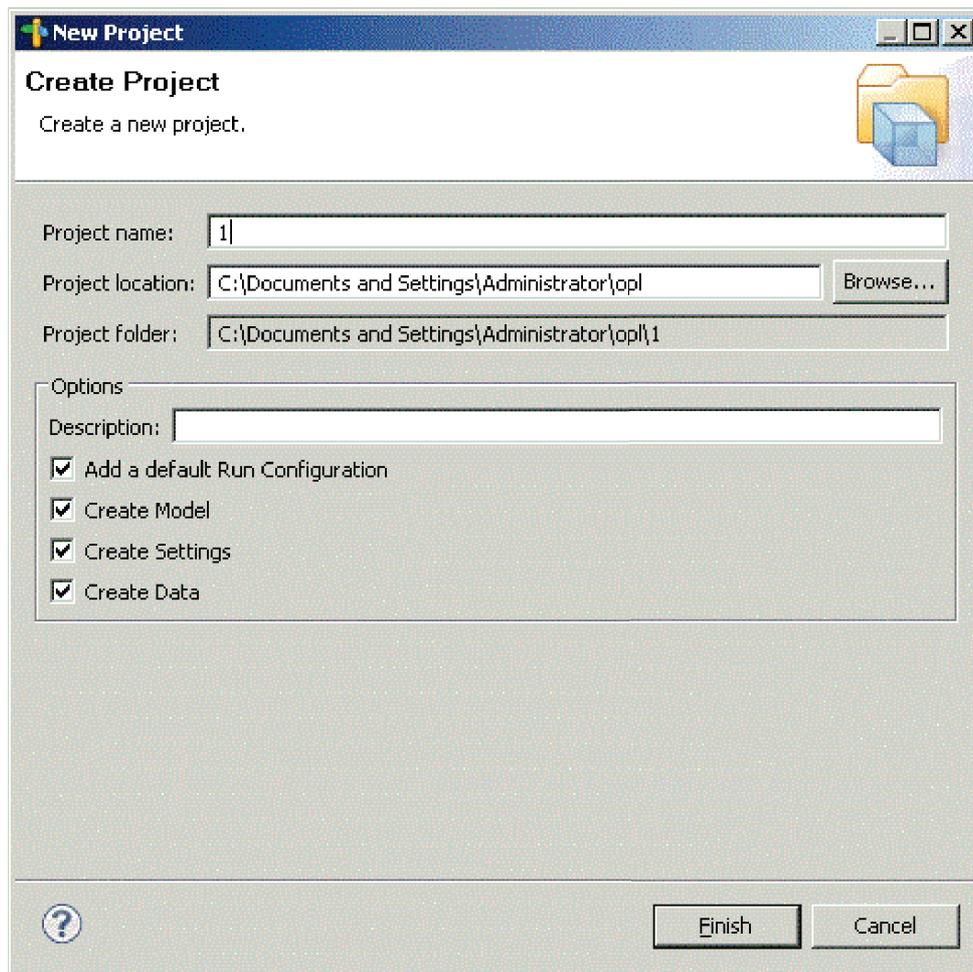


Figura 3: Opciones del proyecto

### 3.2. Resolución básica utilizando CPLEX

La figura 4 muestra la forma más directa de expresar el modelo. Se puede observar que cada una de las ecuaciones del modelo está explícitamente expresada.

```
//Variables continuas positivas
dvar float+ A;
dvar float+ B;
dvar float+ C;

//objetivo
maximize
    50 * A + 40 * B + 30 * C;
```

```

//modelo
subject to {
  recurso1: 0.8 * A + 0.8 * B + 0.3 * C <= 160;
  recurso2: 0.6 * A + 1.2 * B <= 180;
  recurso3: 0.6 * A + 1.0 * B + 0.6 * C <= 110;
  demandaA: A <= 100;
  demandaBMin: 80 <= B;
  demandaBMax: B <= 100;
}

```

Figura 4: contenido del archivo "1.mod"

Lo primero que se puede observar es la sección de variables. En esta sección se indica la existencia de 3 variables A, B y C que representarán la cantidad a producir (y vender) de cada uno de los productos. Es necesario indicar que cada variable es continua y ">= 0" poniendo "float+".

En segundo lugar, está definido el funcional y por último se encuentra la sección de las restricciones. Cada restricción comienza con un rótulo con el cual se podrá identificar la restricción en la solución.

### 3.3. Ejecución y análisis de una corrida

Para realizar la corrida del modelo, se corre su configuración:

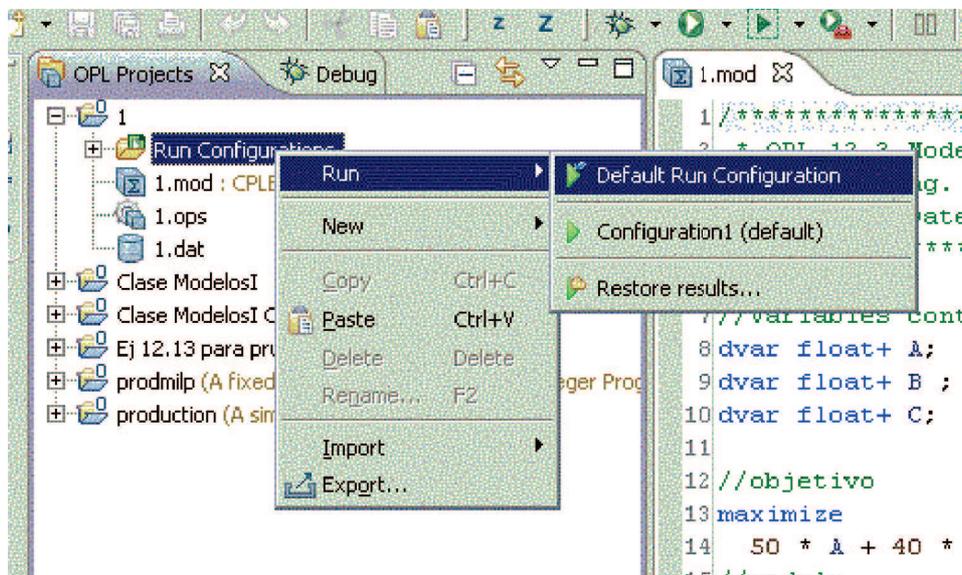


Figura 5: Corrida

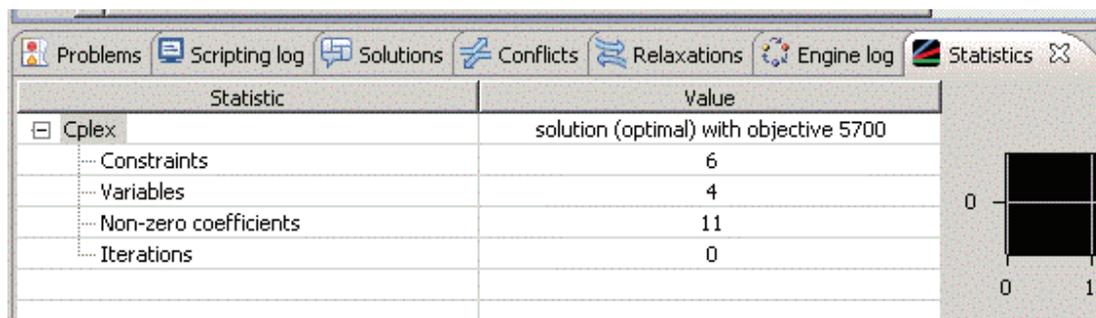
En la solapa "Solution" se obtiene la salida

```
// solution (optimal) with objective 5700
// Quality There are no bound infeasibilities.
// There are no reduced-cost infeasibilities.
// Maximum Ax-b residual = 0
// Maximum c-B'pi residual = 7.10543e-015
// Maximum |x| = 80
// Maximum |slack| = 56
// Maximum |pi| = 83.3333
// Maximum |red-cost| = 20
// Condition number of unscaled basis = 8.2e+000
//
```

```
A = 50;
B = 80;
C = 0;
```

Figura 6: Solapa "Solution"

Se puede encontrar información extra de la corrida en la solapa "Statistics"



Statistic	Value
Cplex	solution (optimal) with objective 5700
Constraints	6
Variables	4
Non-zero coefficients	11
Iterations	0

Figura 7: Solapa "Statistics"

En esta solapa podemos identificar los siguientes elementos.

Cplex: Tipo de solución. En este caso se encontró la solución óptima con valor 5700.

Constraints: Cantidad de filas de la matriz.

Variables: Cantidad de columnas. Existe una columna por cada variable y una por el funcional.

Non-zeros: Cantidad de elementos en la matriz distintos de 0.

Iterations: cantidad de iteraciones requeridas para encontrar el óptimo.

En el “Problem browser” (Window -> Show View -> Problem Browser) se puede ver el estado de las restricciones (Dual, Slack, Bounds), al marcar una restricción aparece el detalle en Properties.

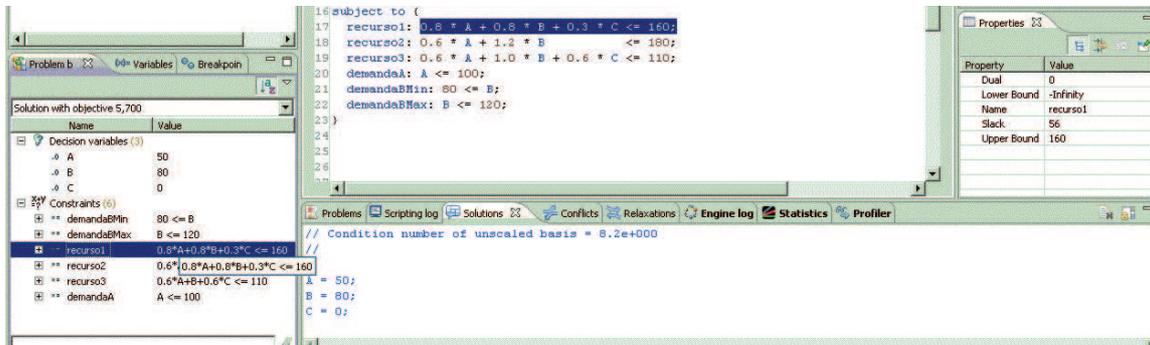


Figura 8: Problem browser

Por ejemplo, podemos ver que la restricción del recurso1, tiene un sobrante de 56 unidades. Es decir, se están consumiendo 104 unidades. La disponibilidad total es de 160. En consecuencia, esta restricción se encuentra inactiva.

Al sobrnarnos recurso1, el valor marginal asociado es 0.

Distinto es el caso de la restricción correspondiente al recurso3, donde se puede ver que la slack vale 0 y su valor marginal es de 83,333. Es decir, si aumentara la cota superior en una unidad (de 110 a 111) ganaremos 83,333 pesos más.

El tercer caso que podemos identificar es el de la restricción de demanda mínima del producto B, demandaBMin. El nivel de actividad de esta restricción es igual a su cota inferior. Si aumentara en una unidad la restricción, estaremos obligando a producir una unidad más y en consecuencia el funcional disminuirá en 43,333.

En la solapa “Solution” se puede identificar el valor de cada variable. La corrida indica que conviene producir 50 unidades de A, 80 unidades de B y ninguna unidad de C.

#### 4. Separación de los datos del modelo

¿Qué pasaría si el día de mañana se decidiera la inclusión de un nuevo producto  $D$ ? Este producto, al igual que los otros, sería procesado por los equipos, con lo cual para cada máquina tendría un determinado tiempo de procesamiento. A su vez, también podría tener una demanda máxima y mínima. Siguiendo nuestra forma de resolverlo en la Figura 1, en cada una de las restricciones de procesamiento tendríamos que agregar el producto  $D$  multiplicado por el tiempo que insume en cada máquina. También agregaríamos las ecuaciones de demanda máxima y mínima. Hacer estos pocos cambios en un modelo muy pequeño como el que estamos analizando es sencillo, ¿Pero qué pasaría si en vez de tener 3 equipos tuviéramos 100?

A grandes rasgos, el problema sobre el cual estamos trabajando tiene 2 tipos de restricciones: el procesamiento de los productos que realiza cada producto y la existencia de demandas máximas y mínimas. Si lo resolviéramos genéricamente, quedaría un modelo como el siguiente:

##### Procesamiento

$$\sum_{\text{Productos}} (\text{Tiempo}_{\text{Producto,Equipo}} * \text{Cant}_{\text{Producto}}) \leq \text{Disp}_{\text{Equipo}} \quad \forall \text{Equipo}$$

##### Demandas mínimas y máximas

$$\text{DemMinima}_{\text{Producto}} \leq \text{Cant}_{\text{Producto}} \leq \text{DemMaxima}_{\text{Producto}} \quad \forall \text{Producto}$$

##### Definición del funcional

$$\text{Max} \rightarrow Z = \sum_{\text{Productos}} (\text{Precio}_{\text{Producto}} * \text{Cant}_{\text{Producto}})$$

Donde  $\text{Tiempo}_{\text{Producto,Equipo}}$ ,  $\text{Disp}_{\text{Equipo}}$ ,  $\text{DemMinima}_{\text{Producto}}$ ,  $\text{DemMaxima}_{\text{Producto}}$  y  $\text{Precio}_{\text{Producto}}$  son constantes.

Figura 9: Modelo genérico

Es importante notar que en el modelo ilustrado en la Figura 9, en ningún momento se hace referencia a productos, equipos, precios o tiempos de procesamiento específicos. Solamente se establece una estructura. Lo que todavía resta sería indicar para que datos queremos resolver el problema. La definición de los datos podría estar dada de la siguiente manera.

	A	B	C
Equipo 1	0,8	0,8	0,3
Equipo 2	0,6	1,2	0
Equipo 3	0,6	1,0	0,6

Equipo	Disponibilidad
Equipo 1	160
Equipo 2	180
Equipo 3	110

Producto	Dem. Máx.	Dem. Mín.
A	100	---
B	120	80
C	---	---

Producto	Precio
A	50
B	40
C	30

Figura 10: Sección de datos

Entonces si volviéramos a la pregunta original, ¿Qué pasaría si en vez de tener 3 equipos tuviéramos 100? Ahora que tenemos los datos separados del modelo, lo único que tenemos que hacer es generar las entradas en las tablas correspondientes. **En ningún momento estaríamos alterando las restricciones del modelo.**

Los mayores beneficios de realizar esta separación de los datos del modelo se pueden ver en otro tipo de problemas, entre ellos los combinatorios. En un problema del viajante si aumentáramos de 100 a 101 ciudades tendríamos que agregar 202 restricciones y modificar otras miles de ecuaciones; si en cambio tuviéramos separado el modelo de los datos lo único que tendríamos que agregar es una fila con las distancias a cada ciudad.

## 5. Separación de datos del modelo en CPLEX

En la siguiente sección se ejemplificará como se hace para separar los datos del modelo en CPLEX. No obstante, debe quedar en claro que la idea de realizar esta separación no es ni propia ni única de CPLEX, otros software como GLPK también lo permiten.

### 5.1. Modelo

Para expresar el modelo genérico, se lo podrá escribir como lo muestra la Figura 11.

Lo primero que se puede observar es la existencia de conjuntos indicados entre llaves, un conjunto estará formado por elementos; en el modelo se podrá iterar sobre estos elementos, pero no hace falta indicar específicamente que elementos son.

De forma similar a la definición de conjuntos se pueden definir arreglos (arrays), estos pueden ser de constantes o de variables, el índice debe ser un conjunto, por ejemplo float PRECIO[PRODUCTOS] son los precios de los productos (constantes) y dvar float+ Produccion[PRODUCTOS] son las cantidades producidas de los productos (variables).

En el funcional se puede ver la aparición de la función sum (sumatoria), a esta función se le pasa como parámetro el conjunto sobre el cual debe iterar.

Podemos ver también la función forall (para todo) que permite repetir una restricción para cada elemento de un conjunto.

Tanto sum, forall como el resto de las funciones de iteración de CPLEX permiten restringir elementos particulares, por ejemplo si queremos definir demandas negativas para elementos sin límite de demanda podemos iterar el forall poniendo “forall(p in CON\_DEMANDA\_MINIMA:DEMANDA\_MINIMA[p] > 0) { ... }”

```

{string} PRODUCTOS = ...;
float PRECIO[PRODUCTOS] = ...;
{string} RECURSOS = ...;
int DISPONIBILIDAD[RECURSOS] = ...;

float CONSUMO[RECURSOS, PRODUCTOS] = ...;
{string} CON_DEMANDA_MINIMA = ...;
{string} CON_DEMANDA_MAXIMA = ...;

int DEMANDA_MINIMA[CON_DEMANDA_MINIMA] = ...;
int DEMANDA_MAXIMA[CON_DEMANDA_MAXIMA] = ...;

//Variables continuas positivas
dvar float+ Produccion[PRODUCTOS];

//objetivo
maximize
    sum(p in PRODUCTOS) Produccion[p] * PRECIO[p];
//modelo
subject to {
    forall(r in RECURSOS) {
        recursos: sum(p in PRODUCTOS) Produccion[p] *
CONSUMO[r,p] <= DISPONIBILIDAD[r];
    }
    forall(p in CON_DEMANDA_MINIMA) {
        demandasMinimas: Produccion[p] >= DEMANDA_MINIMA[p];
    }
    forall(p in CON_DEMANDA_MAXIMA) {
        demandasMaximas: Produccion[p] <= DEMANDA_MAXIMA[p];
    }
}

```

*Figura 11: Contenido del archivo "2.mod"*

## 5.2. Sección de datos

*Nota: Las “,” en los conjuntos y arreglos son opcionales, las pondremos por prolijidad.*

```
PRODUCTOS = {"A", "B", "C"};
PRECIO = [ 50, 40, 30 ];

RECURSOS = {"1", "2", "3"};
DISPONIBILIDAD = [ 160, 180, 110 ];

CONSUMO = [
[0.8, 0.8, 0.3]
[0.6, 1.2, 0.0]
[0.6, 1.0, 0.6]
];

CON_DEMANDA_MINIMA = {"B"};
CON_DEMANDA_MAXIMA = {"A", "B"};

DEMANDA_MINIMA = [ 80 ];
DEMANDA_MAXIMA = [ 100, 120 ];
```

*Figura 12: Contenido del archivo “2.dat”*

## 5.3. Ejecución

Si efectuamos una corrida veremos que el resultado es el mismo aunque los datos se indican de forma diferente, en vez de variables atómicas ahora tendremos arreglos, CPLEX ahora nos mostrará los resultados en tablas para cada arreglo, lo mismo pasa con las restricciones ya que se generaron mediante un iterador.

## **6. Funciones**

CPLEX tiene cuatro funciones de agregación, éstas son: max, min, prod, sum; para agregar restricciones para todo un conjunto se usa forall, aparte de estas funciones básicas CPLEX tiene muchas otras, se puede encontrar la lista completa en la ayuda del IDE: IDE and OPL → Optimization Programming Language (OPL) → Language Quick Reference → OPL keywords → Summary table of OPL keywords.

## 7. Limitando el tiempo de ejecución

Es posible que al intentar resolver problemas de gran escala, el tiempo de ejecución sea de días, meses, años, etc., por esta razón CPLEX nos da la opción de limitar el tiempo de búsqueda de la solución. Para hacerlo, hay que hacer abrir el archivo ".ops" y en Mathematical Programming → General modificar el Global Time Limit que está definido en segundos.

A continuación se muestra una salida en la cual se establecieron 30 segundos como límite de tiempo en un problema de programación lineal mixta con un objetivo de minimización.

```

Tried aggregator 1 time.
MIP Presolve eliminated 140 rows and 49 columns.
Reduced MIP has 2164 rows, 2208 columns, and 10534 nonzeros.
Reduced MIP has 2162 binaries, 0 generals, 0 SOSs, and 0 indicators.
Probing time = 0.02 sec.
Tried aggregator 1 time.
Presolve time = 0.28 sec.
Probing time = 0.02 sec.
Clique table members: 1108.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 2 threads.
Root relaxation solution time = 0.02 sec.

      Nodes
      Node Left      Objective  IInf Best Integer      Cuts/
                                     Best Bound  ItCnt  Gap
      0      0      6951.7105    99
      0      0      15215.4710   96      Cuts: 120   216
      0      0      15260.0695   94      Cuts: 105   275
      0      0      15355.0718  106      Cuts: 66    323
      0      0      17028.5070  108      Cuts: 76    358
      0      0      17665.3149  105      Cuts: 46    392
      0      0      18796.9445   90      Cuts: 36    401
      0      0      18796.9445   89      Cuts: 8     404
*      0+     0                          60954.2245  18796.9445  404  69.16%
*      0+     0                          28372.3734  18796.9445  404  33.75%
*      0+     0                          25368.0420  18796.9445  404  25.90%
*      0+     0                          25331.7973  18796.9445  404  25.80%
      0      2      18796.9445   80      25331.7973  18796.9445  404  25.80%
Elapsed real time = 4.16 sec. (tree size = 0.01 MB, solutions = 4)
*      101    90      integral    0      24623.9947  22691.6823  1774  7.85%
*      123   100      integral    0      24134.7232  22693.9791  1976  5.97%
*      308+  217                          24125.7577  22864.0082  3184  5.23%
      418   268      23357.4861  91      24125.7577  22971.7903  4386  4.78%
*      470+  204                          24121.3202  23100.4961  5974  4.23%
      470   205      23100.4961  42      24121.3202  23100.4961  5974  4.23%
      479   209      23102.3157  92      24121.3202  23102.1693  6091  4.23%
      489   215      23108.3719  84      24121.3202  23105.0075  6220  4.21%

GUB cover cuts applied: 1
Clique cuts applied: 12
Cover cuts applied: 2
Implied bound cuts applied: 4
Flow cuts applied: 1

```

```

Mixed integer rounding cuts applied: 7
Zero-half cuts applied: 14
Gomory fractional cuts applied: 12

Root node processing (before b&c):
  Real time           = 3.70
Parallel b&c, 2 threads:
  Real time           = 26.03
  Sync time (average) = 0.55
  Wait time (average) = 0.00
  -----
Total (root+branch&cut) = 29.73 sec.

```

Figura 13: Engine log con límite de tiempo

Es interesante observar que CPLEX llegó a obtener soluciones enteras, que la mejor antes del límite de tiempo es 24121.3202 y que en el peor de los casos es un 4.21% superior al óptimo; mientras CPLEX progresa en la búsqueda de soluciones enteras va encontrando cotas superiores e inferiores para el óptimo, en un problema de minimización toda solución factible será una cota superior (ya que el óptimo puede ser esa solución u otra con menor valor) y una cota inferior será el óptimo del mismo problema sin forzar a las variables a ser enteras (como se quitan restricciones el óptimo será ese u otro con mayor valor); teniendo estas dos cotas podemos saber a qué distancia máxima estamos del óptimo real, esto lo indica el "Gap".

CPLEX muestra la evolución de las soluciones encontradas en la solapa "Statistics", Best Node es la mejor solución continua, Integer solution una solución entera encontrada y Best Integer la evolución de la mejor solución entera.

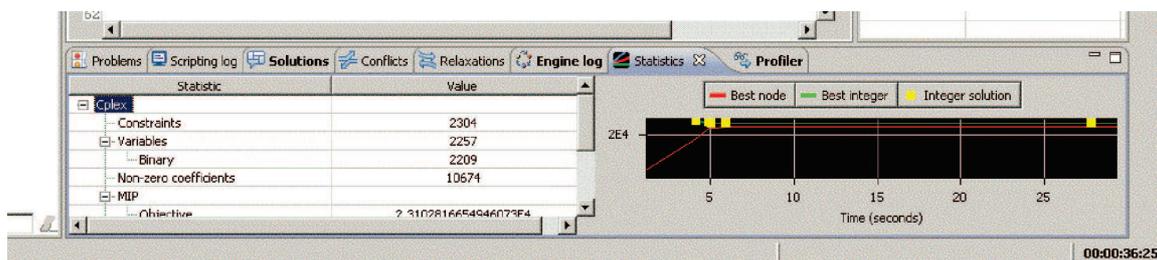


Figura 14: Evolución de las soluciones