27 Congreso Nacional de Estadística e Investigación Operativa
Lleida, 8–11 de abril de 2003

DESARROLLO DE UNA HERRAMIENTA DE OPTIMIZACIÓN BINIVEL UTILIZANDO CPLEX Y MATLAB Y GENERACIÓN DE PROBLEMAS DE PRUEBA

A. González Uzábal, C. Galé[1], y J. Campos[2],

[1]Departamento de Métodos Estadísticos
Universidad de Zaragoza, 50018 Zaragoza, España
E-mail: cgale@posta.unizar.es
[2]Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, 50018 Zaragoza, España
E-mail: jcampos@posta.unizar.es

RESUMEN

En este trabajo se presenta el desarrollo de una herramienta para resolver problemas de optimización binivel y en particular de un nivel, tanto con objetivos lineales como lineales fraccionarios, implementada en CPLEX® y en MATLAB®. La herramienta permite seleccionar entre varios algoritmos de resolución y obtener medidas de evaluación del rendimiento de cada uno.

Se presenta también un método para la generación automática de problemas binivel de prueba que ayude a evaluar este tipo de herramientas. Mediante una serie de transformaciones de la matriz de coeficientes tecnológicos y la función objetivo de segundo nivel, se asegura la factibilidad del problema y que la solución del problema binivel no sea la trivial.

Palabras y frases clave: Optimización binivel fraccionaria, Test de problemas, Herramienta.
Clasificación AMS: 68N01 90B50 90C32 90C49.

1 Introducción

La programación lineal (Bazaraa et al (1990)) es una de las técnicas de modelización y resolución de problemas más utilizadas en la Investigación Operativa. Se aplica en problemas de toma de decisiones en los que la función objetivo es lineal y la región de factibilidad es un poliedro.
Los modelos de programación multinivel dividen el control de las variables de decisión entre niveles ordenados dentro de una estructura de planificación jerárquica.

Un planificador a un nivel de la jerarquía puede tener su función objetivo y conjunto de decisiones factibles determinado en parte por otros niveles. Sin embargo, sus instrumentos de control pueden permitirle influir en las políticas a otros niveles y a partir de ahí, mejorar su propia función objetivo.
Los procesos de toma de decisiones que pueden representarse adecuadamente mediante un modelo de programación multinivel, tienen las siguientes características comunes:

- El sistema está compuesto de unidades de toma de decisiones que interactúan dentro de una estructura jerárquica de niveles.
- Cada nivel inferior selecciona su política de acción después de conocer las decisiones de los niveles superiores. Los niveles de decisión optimizan sus objetivos de forma secuencial.
- Cada unidad de decisión intenta optimizar su objetivo independientemente de las otras, si bien puede resultar alterada por las acciones y reacciones de alguna de ellas.
- La influencia de un nivel superior en el problema correspondiente a un nivel inferior se puede reflejar tanto en su función objetivo como en su conjunto de decisiones factibles.

La herramienta que se describe aborda problemas con dos niveles de decisión, a menudo con intereses diferentes.
En relación con los algoritmos diseñados para resolver el problema binivel, se han considerado algoritmos enumerativos, principalmente en el problema binivel lineal. Entre estos algoritmos están los desarrollados en Bialas et al (1984) y Calvete et al (1999). Cuando el problema del nivel inferior se reemplaza por sus condiciones de Karush-Kuhn-Tucker, y aprovechando la naturaleza disyuntiva de las condiciones de la holgura complementaria, se consideran algoritmos de ramificación y acotación y algoritmos de pivote complementario (Judice et al (1992), Hansen et al (1992)). Cuando el problema binivel incluye funciones no lineales se han desarrollado algoritmos que incluyen la determinación de direcciones de descenso (Savard (1994)), la consideración de funciones de penalización (White (1997)) o la utilización de técnicas de optimización global (Tuy et al (1994)). Generalmente, estos algoritmos garantizan exclusivamente la optimalidad local de la solución encontrada.
En particular los problemas de programación binivel, consideran dos niveles de decisión, nivel superior o primer nivel y nivel inferior o segundo nivel. El nivel superior ha de tomar sus decisiones teniendo en cuenta la repercusión que van a tener sobre el nivel inferior y la reacción de éste ante la decisión de aquél. Los dos niveles optimizan sus objetivos de forma secuencial y sus variables de decisión se mueven conjuntamente en el conjunto de restricciones. La formulación general de este tipo de problemas es:

$$\min_{x_1} f_1(x_1, x_2)$$
donde x_2 es solución de
$$\min_{x_2} f_2(x_1, x_2)$$
sujeto a: $(x_1, x_2) \in S$

El trabajo se ha organizado como sigue. En primer lugar, se explican las características principales de la herramienta desarrollada y los problemas a los que se puede aplicar. En la sección 3 se trata la generación de una batería de pruebas para evaluar el rendimiento de la herramienta de optimización binivel propuesta. A continuación, se muestran los resultados de la evaluación de la herramienta y el generador de problemas de prueba. Finalmente, se presentan algunas conclusiones derivadas de este estudio.

2 Herramienta de optimización binivel

2.1 Capacidades y descripción general

El objetivo de la herramienta es el de proporcionar una plataforma de cálculo y gestión para problemas de optimización binivel aplicando MATLAB® y CPLEX®. La herramienta trabaja con cuatro formatos de ficheros:

- *.mat* es el formato de datos de MATLAB. Los problemas introducidos se convierten a este formato antes de proceder a su resolución.
- *.lp2* es un formato de texto desarrollado especialmente para problemas binivel a partir del formato *.lp* que se utiliza en aplicaciones como LINDO®, lpsolve® ó CPLEX.
- *.3lp* es un formato de texto desarrollado especialmente para facilitar la gestión de los problemas binivel usando CPLEX.
- *.txt* es un formato de texto simple donde se almacena la salida del generador de problemas de prueba.

La herramienta permite:

- Introducir un problema a través del teclado o a través de formatos de texto. Los problemas introducidos se almacenan en formatos *.3lp* y *.mat*.
- Modificar un problema mediante los formatos de texto *.lp2* y *.3lp*.
- Resolver un problema. Se elige el problema a cargar, el algoritmo de resolución, el grado de verbosidad, la precisión de cálculo y el número máximo de iteraciones.

- Traducir los formatos de los ficheros, mediante las siguientes herramientas de traducción.
  - de *.txt* a *.mat* para cargar en MATLAB un problema generado.
  - de *.txt* a *.3lp* para cargar en CPLEX un problema generado.
  - de *.lp2* a *.mat* y de *.mat* a *.lp2* dando la posibilidad de manejar los ficheros de MATLAB en formato de texto.

2.2 Algoritmos implementados

Existen herramientas bien conocidas y citadas en la literatura para resolver problemas de programación matemática de un solo nivel, sin embargo los problemas binivel son un campo mucho menos explorado. En este apartado se enumeran las técnicas de optimización implementadas en la herramienta desarrollada y se proporciona una visión general de sus características. En primer lugar se presentan los dos algoritmos implementados para la resolución de problemas de un sólo nivel y a continuación se presentan el algoritmo *K-ésimo mejor punto extremo* (Bialas et al (1982)) y el algoritmo *Búsqueda de Bases de Interés* (Calvete et al (1999)), que consideran problemas binivel.
Por tanto, la herramienta desarrollada incluye la implementación de los siguientes algoritmos.

- Algoritmo Simplex (Bazaraa et al (1990))
  Se ofrece, como parte de la herramienta, la versión del algoritmo simplex que proporcionan MATLAB y CPLEX. El método simplex es un método numérico de resolución muy extendido para problemas de programación lineal de un solo nivel, con la forma:

$$\begin{aligned} &\text{Minimizar } c^1 x_1 + c^2 x_2 \\ &\text{sujeto a:} \\ &\quad A_1 x_1 + A_2 x_2 \leq b \\ &\quad x_1, x_2 \geq 0 \end{aligned}$$

- Transformación de Charnes Cooper en problemas de programación fraccionaria lineal (Bazaraa et al (1994))
  La transformación de Charnes y Cooper consiste en realizar un cambio de variable que permite resolver problemas de programación fraccionaria lineal de un sólo nivel a partir de la resolución de un problema de programación lineal. La formulación de un problema fraccionario es:

$$\begin{aligned} &\text{Minimizar } \frac{c^{11} x_1 + c^{12} x_2}{c^{21} x_1 + c^{22} x_2} \\ &\text{sujeto a:} \\ &\quad A_1 x_1 + A_2 x_2 \leq b \\ &\quad x_1, x_2 \geq 0 \end{aligned}$$

- Algoritmo K-ésimo mejor (Bialas et al (1982))
  El algoritmo K-ésimo mejor es un algoritmo de resolución general para problemas de programación binivel, cuya solución óptima está garantizado que se alcanza en un punto extremo de la región de factibilidad común a ambos niveles de decisión. Se propuso por primera vez para resolver el problema binivel lineal, cuya formulación matemática es:

  $$\begin{aligned}&\text{Minimizar } k^1x_1 + k^2x_2\\&\text{donde dado } x_1,\ x_2 \text{ es solución de:}\\&\quad\text{Minimizar } c^1x_1 + c^2x_2\\&\quad\text{sujeto a:}\\&\qquad A_1x_1 + A_2x_2 \leq b\\&\qquad x_1, x_2 \geq 0\end{aligned}$$

  Este algoritmo obtiene en primer lugar la solución óptima del primer nivel. Comprueba si es factible binivel y si no lo es, busca la siguiente mejor solución para el primer nivel. De nuevo comprueba si ésta es factible binivel y si no lo es, busca la siguiente, y así hasta encontrar la k-ésima mejor solución, que sea la primera factible binivel.

- Algoritmo de Búsqueda de Bases de Interés (Calvete et al (1999))
  Este algoritmo fue desarrollado específicamente para problemas de programación binivel lineal/lineal fraccionario si bien es aplicable a los problemas binivel lineales por ser un caso particular de aquéllos. El algoritmo de Búsqueda de Bases de Interés (BBI) realiza un examen restringido de submatrices (bases) de la matriz de coeficientes tecnológicos, correspondiente a las variables del segundo nivel. Este examen permite encontrar una solución óptima global en un número finito de pasos. El algoritmo busca características adecuadas para que una base sea solución y se mueve de base en base desechando aquéllas que no cumplan las características determinadas, a la vez que determina nuevas condiciones para la base solución.
  El problema que se trata de resolver tiene esta forma:

  $$\begin{aligned}&\text{Minimizar } k^1x_1 + k^2x_2\\&\text{donde dado } x_1,\ x_2 \text{ es solución de:}\\&\quad\text{Minimizar } \frac{\alpha + c^{11}x_1 + c^{12}x_2}{\beta + c^{21}x_1 + c^{22}x_2}\\&\quad\text{sujeto a:}\\&\qquad A_1x_1 + A_2x_2 \leq b\\&\qquad x_1, x_2 \geq 0\end{aligned}$$

3 El generador de problemas de prueba

En la literatura, hasta el conocimiento de los autores, no se ha presentado ningún programa informático que proporcione problemas de programación binivel, suficien-

temente general, para evaluar el funcionamiento de herramientas como la desarrollada en este trabajo. Por este motivo, se ha implementado un generador de problemas de programación binivel lineal/lineal fraccionarios. El desarrollo teórico del algoritmo está fundamentado en el análisis de los requerimientos planteados por este tipo de problemas, y el análisis realizado por Moshirvaziri et al (1996) y Calamai et al (1993), para generar problemas de programación binivel lineal y lineal/cuadráticos, respectivamente.
Todos los coeficientes del problema, tanto de las funciones objetivo como de las restricciones, se han generado de forma aleatoria. El vector de recursos se construye de la siguiente forma:

$$b_i = \sum_{j=1}^{n} c_{ij}$$

Además, para asegurar la acotación del problema, la última restricción del problema tendrá esta forma:

$$c_1 x_1 + \ldots + c_n x_n \leq \sum_{i=1}^{n} c_i \qquad \text{donde } c_1, \ldots, c_n \geq 0$$

Por otro lado, el generador resuelve los problemas relajados, correspondientes al primer nivel para asegurar que la solución del problema no es la trivial, en cuyo caso, modifica los coeficientes de la función objetivo, atendiendo al vector gradiente, para desplazar la solución óptima a otros puntos de la región factible.
A continuación, se muestra una descripción en pseudocódigo del algoritmo generador.

```
generamos semilla aleatoria
    mientras no generado
        aleatorio(Objetivo1, Restricciones)
        si tipoproblema = lineal
            Objetivo2 = -Objetivo1
        sino
            aleatorio(alfa, Objetivo2Numerador)
            aleatorio(beta, Objetivo2Denominador)
            mientras que no encontrado
               [minimo, maximo] = resolver(Objetivo2, Restricciones)
                si (minimo > -beta) ó (maximo < beta)
                    encontrado = 1
                sino
                    si superado numero de intentos
                        aleatorio(Objetivo2)
                    sino
                        incrementar(beta)
                    fsi
```

```
            fsi
        fmq
    fsi
mientras que iguales
        solucionPrimerNivel = simplex(Objetivo1, Restricciones)
        solucionSegundoNivel = simplexAcotado(Objetivo2, Restricciones)
        iguales = comparar(solucionPrimerNivel, solucionSegundoNivel)
        si iguales
            variar(Objetivo1, componente a variar)
            adaptar(Objetivo1, Objetivo2)
            si (productoEscalar(Objetivo1, Objetivo1Anterior) < 0) y
                (variadas todas las componentes)
                aleatorio(Objetivo1)
                adaptar(Objetivo1, Objetivo2)
            fsi
        fsi
    fmq
fmq
almacenar los valores en el fichero de destino
```

En el código anterior se han considerado las siguientes convenciones:

`aleatorio(estructura, estructura2)` Rellena las estructuras de datos con valores aleatorios generados a partir de una distribución uniforme, tantos como sea necesario. El rango de valores de la distribución determinará las dimensiones de los coeficientes del problema y puede ser elegido por el usuario.

`comparar(solucion, solucion2)` Devuelve `cierto` o `falso` según sean, iguales o no, componente a componente las soluciones obtenidas.

`resolver(funcion, restricciones)` Devuelve la solución máxima y mínima calculadas con el método simplex de una `funcion` objetivo sujeto a un conjunto de `restricciones`.

`incrementar(parametro)` Aumenta el valor de `parametro` en un orden de magnitud.

`adaptar(objetivoLider, objetivoSeguidor)` Transforma `objetivoSeguidor` con relación a `objetivoLider` para tratar de obtener las soluciones más alejadas.

4 Evaluación de los desarrollos

En esta sección se describe la evaluación de las diferentes partes de la herramienta desarrollada. En primer lugar, de los diferentes formatos de ficheros con los que

trabaja, a continuación de la generación de problemas de prueba mediante el algoritmo presentado en este trabajo, finalmente el rendimiento de los algoritmos de optimización implementados en la herramienta. La plataforma sobre la que se han realizado las pruebas es un *Intel Pentium® III* a 1000 MHz y con sistema operativo *Windows Millenium®*.

4.1 Almacenamiento de problemas en ficheros

La tabla 1 muestra el coste de almacenamiento en espacio de los diferentes formatos de ficheros utilizados para distintos tamaños de problema.

Tamaño del Problema	*.mat*	*.lp2*	*.3lp*	*.txt*
2 x 6	2K	1K	(1+1+1)K	1K
20 x 8	2K	3K	(1+1+1)K	3K
100 x 80	19K	127K	(60+84+7)K	93K

Tabla 1: Coste en espacio de los formatos de fichero utilizados.

Si bien con ficheros pequeños no hay apenas diferencias de costes, el formato de datos de MATLAB se muestra claramente más solvente para los problemas más grandes. Los otros formatos se mantienen en un coste espacial de

$$O(nv * nr * tc)$$

siendo nv el número de variables del problema, nr el número de restricciones y tc el tamaño ocupado por la representación del máximo coeficiente permitido en el problema.

4.2 Generación de problemas

En la evaluación del generador de problemas de prueba se han tomado los tiempos de ejecución en la generación de un problema bajo diferentes condiciones. Se han considerado un número de variables entre 20 y 100 y para cada uno de estos valores, el 25% o el 50% del total de variables son controladas por el segundo nivel. El número de restricciones es, en todos los casos, del 40% del número de variables. Para cada combinación de los valores se han realizado 10 pruebas. Se ha recogido el tiempo de generación del mismo en segundos. La tabla 2 muestra el valor de tiempo medio en segundos y los valores máximo y mínimo. Como se muestra en las figuras 1 y 2 se aprecia una clara diferencia en el tiempo de generación de los problemas, dependiendo del número de variables controladas por el segundo nivel. Esto viene derivado de la relación directa entre el número de variables de nivel inferior y el grado de libertad que tiene el generador para asegurar una solución no trivial.
Los problemas pequeños son generados de forma prácticamente instantánea, sin embargo, en problemas con más de 40 variables y un porcentaje bajo de variables de segundo nivel, el coste medio se incrementa. La diversidad de valores en el coste

Número de Variables	% Variables 2 Nivel	Tiempo Medio(s)	Tiempo Mínimo	Tiempo Máximo
20	25%	1.4	1	5
20	50%	1.6	1	4
40	25%	11.0	1	45
40	50%	2.4	2	4
60	25%	46.3	3	181
60	50%	2.4	2	4
80	25%	46.2	3	129
80	50%	4.0	3	6
100	25%	77.6	11	126
100	50%	6.6	5	10

Tabla 2: Coste temporal de la generación de problemas.

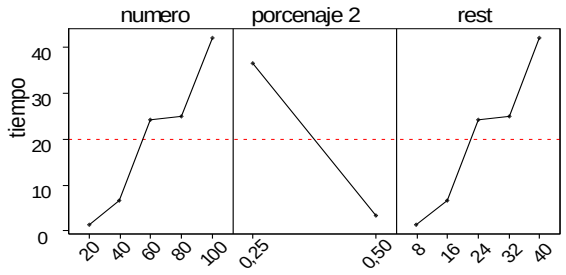

Figura 1: Efectos principales

temporal se debe a la generación aleatoria de los problemas, lo que implica que el grado de adecuación de los primeros valores que se obtengan condicionará los caminos de ejecución del algoritmo. En cualquier caso el generador proporciona problemas de prueba relativamente grandes en un periodo de tiempo del orden de minutos.

4.3 Resolución de problemas

La evaluación de la herramienta de optimización implementada se ha realizado a partir de la toma de los tiempos de ejecución de problemas generados automáticamente, planteando distintos escenarios según los valores de los factores: número de variables global del problema, porcentaje de variables asignadas al segundo nivel y número de restricciones comunes a ambos niveles de decisión.
Se ha trabajado con un número de variables entre 10 y 90, la proporción de variables de segundo nivel se ha movido entre un 10% y un 50% del número total de variables y el número de restricciones ha variado entre el 20% y el 60% sobre el número de variables. De esta forma hemos contemplado escenarios con muchas y pocas variables, con altos grados de control del segundo nivel o gran determinación del primero y con escasez o no de restricciones.

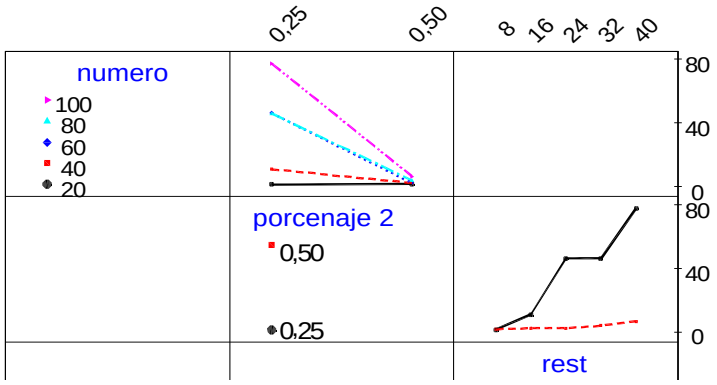

Figura 2: Interacciones

Esto permitirá formular hipótesis sobre la influencia del número de restricciones en el rendimiento de los algoritmos. Además, el alto rango de variables analizado permite comprobar la evolución del coste temporal conforme aumenta el número de variables. Cada algoritmo se ha probado con un límite de 100 iteraciones, ofreciéndose en caso de no ser suficientes para encontrar la solución óptima, el porcentaje de adecuación a esa solución. En la tabla 3 se muestra que:

- La implementación del algoritmo K-ésimo mejor en CPLEX obtiene, en general, peores resultados que en MATLAB, especialmente en los problemas de menor tamaño, siendo la diferencia relativamente pequeña. Esto se puede deber al mayor coste de gestión de matrices en CPLEX que en MATLAB, especialmente en problemas pequeños. En los problemas grandes las rutinas más depuradas de CPLEX parecen compensar en parte el gasto de gestión de las matrices.

- La implementación del algoritmo BBI es más eficiente, en general, en CPLEX que en MATLAB. Las diferencias llegan a ser de un orden de magnitud, lo que puede derivarse del soporte que da CPLEX a la programación entera.

- Aunque la implementación del algoritmo BBI parece sufrir más con problemas con un 60% de restricciones, donde no llega a encontrar la solución óptima en el máximo de iteraciones, en tiempos relativamente cortos ofrece soluciones con un grado de adecuación superior al 90%.

- El número de variables del segundo nivel es un factor muy influyente en los tiempos de ejecución. La mayor libertad del segundo nivel, al aumentar el número de variables que controla, complica la localización del punto extremo óptimo.

- El porcentaje de restricciones influye en los tiempos de ejecución principalmente cuando el número de variables del segundo nivel se incrementa. Esto

Número de variables	% var. de el nivel 2	% restricc. respecto a variables	K-ésimo MATLAB	BBI MATLAB	K-ésimo CPLEX	BBI CPLEX
10	10%	20%	0.06	0.11	0.71	0.11
10	10%	40%	0.05	0.11	1.15	0.11
10	10%	60%	0.06	0.22	0.77	0.06
10	30%	20%	0.55	0.61	0.93	0.44
10	30%	40%	0.61	0.60	0.99	0.88
10	30%	60%	0.61	6.59	0.77	2.80
10	50%	20%	0.05	0.28	1.37	0.33
10	50%	40%	0.11	0.16	0.60	2.04
10	50%	60%	0.77	31.64^1 51.7%	0.77	32.35^1 51.7%
20	10%	20%	0.16	0.93	0.94	0.66
20	10%	40%	0.16	0.44	1.31	0.06
20	10%	60%	0.16	2.64	1.43	0.39
20	30%	20%	3.24	48.44	1.09	5.66
20	30%	40%	1.93	21.23^1 95.2%	2.09	3.9^1 39.5%
20	30%	60%	0.66	20.7	0.94	4.39
20	50%	20%	0.28	14.23^1 81.9%	1.04	3.57
20	50%	40%	1.46	15.49^1 21.6%	1.2	4.12^1 57.9%
20	50%	60%	35.2^1 19.1%	16.97^1 22.3%	6.59^1 65.0%	4.23

(1) $A^1B\%$, % de adecuación a la mejor solución obtenida y tiempo invertido en obtenerla (100 iteraciones).

Tabla 3: Tiempos de ejecución en segundos

viene dado especialmente por el aumento que genera en las matrices de datos lo que genera un mayor coste de gestión.

La figura 3 y la figura 4 confirman, a partir de los datos de la tabla 3, los efectos de los factores estudiados en relación con el tiempo de ejecución, así como las interacciones que se producen entre dichos factores.
En la tabla 4 se incluyen los tiempos de ejecución para nueve resoluciones con distintos tamaños de problema, aumentando progresivamente el número de variables. Analizando dicha tabla se observa lo siguiente.

- Hasta 50 variables el algoritmo k-ésimo mejor tiene un rendimiento bueno, produciéndose para problemas de más de 60 variables un aumento dramático del coste temporal.

- Al aumentar el número de variables de los problemas, mayor es la diferencia a favor del algoritmo BBI. Se llegan a obtener diferencias temporales de dos órdenes de magnitud, consiguiendo además una solución más próxima a la

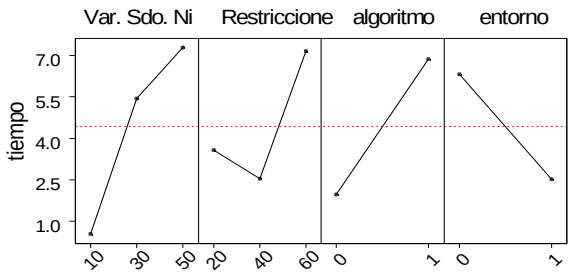

Figura 3: Efectos principales

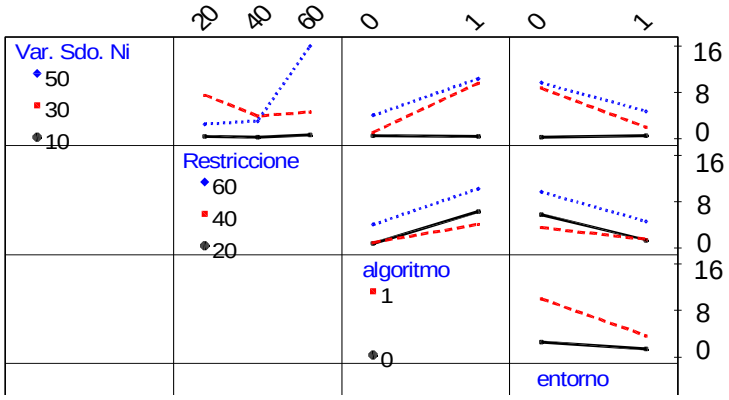

Figura 4: Interacciones

óptima. Por tanto, el algoritmo K-ésimo mejor sufre mucho más el efecto del tamaño de los problemas a resolver y el algoritmo de BBI parece mostrarse más estable para los tamaños extremos.

- El resto de comentarios es análogo a los referidos en la tabla 3

5 Conclusiones

En este trabajo se ha presentado el desarrollo de una herramienta para resolver problemas de optimización binivel, en particular, de un nivel, tanto con objetivos lineales como lineales fraccionarios, implementada con CPLEX y con MATLAB. En la evaluación de la herramienta se ha comparado las posibilidades de ambas plataformas de programación y el rendimiento de dos algoritmos de resolución de problemas binivel lineales y lineales/lineales fraccionarios.
En la evaluación de la herramienta de optimización se ha precisado la generación automática de problemas binivel de prueba, para lo cual, se ha desarrollado una aplicación que permite obtener dichos problemas. Además, se ha analizado el coste en tiempo de ejecución dependiendo del tamaño de los problemas.

Número de variables	% var. de nivel 2	% restricc. respecto a var.	K-ésimo MATLAB	BBI MATLAB	K-ésimo CPLEX	BBI CPLEX
30	30%	20%	1.76	34.28	8.68	5.33^1 99.3%
40	30%	20%	33.23	18.92	27.57	3.57^1 53.6%
50	30%	20%	2.41	25.82	15.98	4.56^1 96.3%
60	30%	20%	424.08^1 20.6%	20.65^1 41.1%	265.62^1 72.4%	5.11
70	30%	20%	637.29^1 62.5%	26.08^1 48.6%	259.45^1 68.7%	13.24
80	30%	20%	724.00^1 68.2%	40.53^1 92.9%	289.29^1 67.6%	13.89
90	30%	20%	1556.35^1 86.1%	40.53^1 76.9%	1761.27^1 98.8%	3.63

(1) A^1B%, % de adecuación a la mejor solución obtenida y tiempo invertido en obtenerla (100 iteraciones).

Tabla 4: Tiempos de ejecución según el número de variables.

6 Agradecimientos

Este trabajo se ha financiado parcialmente en el ámbito del Proyecto TIC2002-04334-C03-02 del Ministerio de Ciencia y Tecnología.

Referencias

Bazaraa M.S, Jarvis J.J, Sherali H.D, (1990). Linear programming and network flows, John Wiley and Sons.

Bazaraa M.S, Shety, Sherali H.D, (1994). Nonlinear Programming, Theory and Applications, John Wiley and Sons.

Bialas W.F, Karwan M.H, (1982), Two Level Linear Programming, *Management Science 30, 1004–1024*

Calvete H.I, Galé C, (1999), The bilevel linear/linear fractional programming problem, *European J. of Operational Research 114(1), 188–197*

Calamai P.H, Vicente L.N, (1993), Generating linear and linear-quadratic bilevel programming problems, *SIAM J. Scientific and Statistical Computing 14*

Dyer M.E, Proll L.G (1977), An Algorithm for Determining all Extreme Points of a Convex Polytope, *Mathematical Programming 12, 81-96*

Hansen P, Jaumard B. y Savard G, (1992) New branch-and-bound rules for linear bilvel programming *SIAM Journal on Scientific and Statistical Computing, 13, (5), 1194-1217*

Júdice J, and Faustino A. M, (1992) A sequential LCP method for bilevel linear programming, *Annals of Operations Research 34, 89-106*

Moshirvaziri K, Amourzegar M.A y Jacobsen S.E, (1996), Test problem construction for Linear Bilevel Programming Problems, *Journal of Global Optimization 8: 235-243*

Savard G. y Gauvin J, (1994) The Steepest Descent Direction for the Nonlinear Bilevel Programming Problem *Operations Research Letters 15, 265-272*
Tuy H, Migdalas A, y Värbrand P, (1994) A quasiconcave minimization method for solving linear two-level programs *Journal of Global Optimization*4243-263
White D.J, (1997) Penalty Function Approach to Linear Trilevel Programming *J. of Optimization Theory and Applications 93 (1) 183-197*
