10-6-2020

PROGRAMACIÓN LINEAL
Problema del legado

Algoritmia Básica

Néstor Monzón González
735418

1 – Formalización del problema

Los datos del problema son los siguientes:

- m bienes.
- n herederos.
- v_i valor en euros de cada bien (con $i \in [1, m]$).

El objetivo es repartir los m bienes entre los n herederos, minimizando la diferencia entre la máxima suma de valores heredados y la mínima.

Sea b_j el valor de los bienes heredados por el heredero j (con $j \in [1, n]$). Podemos formular el objetivo como:

$$\min \max |b_j - b_k|$$

Para todo $j \in [1, n-1]$, $k \in [j+1,\ n]$. Para evitar el min-max, se puede transformar en:

$$\min(maxb - minb)$$

Donde $maxb$ es el máximo beneficio ($maxb = \max_j b_j$) y $minb$ el mínimo ($minb = \min_j b_j$).

Finalmente, para definir el reparto concreto de bienes, necesitamos, al menos, una variable para cada bien que determine a qué heredero se asigna. En lugar de optar por una variable entera por bien cuyo valor sea el índice de este heredero, se ha considerado más simple utilizar variables binarias.

Así, usaremos n*m variables binarias X, donde $x_{ij} = 1$ sii el heredero i hereda el bien j. Todas las variables de un mismo valor j deben sumar 1, ya que solo uno puede heredar ese bien.

Primera instancia

Se ha formalizado una instancia muy simple del problema para comprobar que su solución sea la correcta, indicando que su formalización también lo es:

- Problema (en $mini\_prob.txt$):

$Problema\ con\ n = 2, m = 4$

$Valores$: 549,669,926,978

La solución intuitiva (ya que los valores son muy similares) es asignar el máximo y el mínimo al heredero 1 y los otros dos al 2: $b1 = 978 + 549$, $b2 = 926 + 669$.

- En CPLEX (simplificado por espacio), lo formalizamos como:

$minimize\ maxb - minb$
$subject\ to$
$maxb >= b1$
$maxb >= b2$

$minb <= b1$
$minb <= b2$
$b1 = 549 * x11 + 669 * x12 + 926 * x13 + 978 * x14$
$b2 = 549 * x21 + 669 * x22 + 926 * x23 + 978 * x24$
$x11 + x21 = 1$
$x12 + x22 = 1$
$x13 + x23 = 1$
$x14 + x24 = 1$
$generals$
$x11\ x12\ x13\ x14\ x21\ x22\ x23\ x24\ binary$
end

Así, tenemos las primeras 4 (en general, $2n$) restricciones para las variables maxb, minb; las siguientes 2 (n) para los valores de b; y las últimas 4 (m) para los x. La solución obtenida en el optimizador de la web de *Linear Optimization Solver*, implementado a partir del Glpk.js - GNU Linear Programming Kit for Javascript es, efectivamente, la misma que la que habíamos propuesto. También se ha resuelto con el programa final, llevando a la misma solución ($mini\_prob\_solucion.txt$).

2 – Generación de problemas

Por otra parte, para obtener automáticamente múltiples instancias del problema, se ha desarrollado el script $generar\_problemas.py$. Se ha seleccionado representar la información de una serie de p problemas en un fichero de texto, para permitir contener el número de problemas necesario por cada fichero.

La sintaxis del fichero será la siguiente:

- Primera línea: número de problemas p.

A continuación, p bloques con las siguientes líneas, para cada problema:

- Primera: dos números n y m, separados por un espacio
- Siguiente: m números que representan cada valor v_i en orden, separados por un espacio.

Se ha decidido representar los valores v_i como enteros. En el enunciado, solo se especifica que los valores son en euros. Si se necesita tener en cuenta los céntimos, es trivial representar los valores en esa unidad. Aunque, generalmente, los solucionadores son más rápidos con problemas con variables reales, en este caso las variables de decisión x son binarias, con lo que ya sería un problema mixto. Así, y para no llevar a errores por cálculos con coma flotante, se ha decidido modelar el problema como puramente entero.

Los parámetros del generador serán el número de problemas y el nombre del fichero de salida. Opcionalmente, se podrán añadir los rangos del número de herederos, de bienes y de los valores. En general, se invoca de la siguiente forma:

$generar\_problemas.py\ <salida.txt>\ [num\_problemas]\ [<\min\ \_n>,<\max\ \_n>]$

$\ldots\ [<min\_m>,<max\_m>]\ [<min\_v>,<max\_v>]$

3 – Integración de la herramienta de PL

Tras una comparación de las herramientas de programación lineal disponibles con APIs de Python, se ha seleccionado CPLEX (de IBM) principalmente por su popularidad, que lleva a un mayor número de guías y ayuda disponibles en internet.

Para modelar cada instancia del problema se ha creado la clase $Problema$ (en $problema.py$), que contiene los datos descritos en el primer apartado. Se compone principalmente de dos métodos: el constructor, que lo inicializa a partir de un fichero (con la sintaxis descrita en el apartado 2), y $solucionar()$, que computa la solución a través de cplex, la guarda, y mide el tiempo que le lleva.

También contiene otros métodos auxiliares, como el que guarda la solución y el tiempo en un segundo fichero de texto para futuro análisis. La sintaxis de este fichero es la siguiente: en cada línea, se almacena el valor de las variables importantes de un problema y su solución, separadas por espacios. Todas las variables son enteras, excepto t, el tiempo de ejecución en segundos. Cada línea es pues:

$n\ m\ t\ (maxb - minb(f\ objetivo))\ x11 \ldots xnm\ b1 \ldots bn\ maxb\ minb$

En $solucionar()$ se ha seguido el esquema general del ejemplo de Github example.py, aunque adaptándolo para este problema y, principalmente, generalizando las variables y demás parámetros para cualquier instancia, en función de n, m y los valores v. Esto se ha implementado con comprensión de listas.

En cuanto a la gestión del tiempo de ejecución, se ha establecido un límite de 5 minutos (para permitir resolver un gran número de problemas sin que uno lleve horas, como $prob\_excesivo.txt$, que terminó en 14 horas con solo $n = 10$ y $m = 50$). Además, aunque la propia librería de cplex provee una medición del tiempo de resolución por salida estándar, se ha decidido medir externamente al comprobar que con la llamada a la función este tiempo era, generalmente, ligeramente superior al de cplex.

Para más detalles sobre la implementación, se pueden consultar los comentarios de $problema.py$, que explican cada uno de los parámetros introducidos al solucionador.

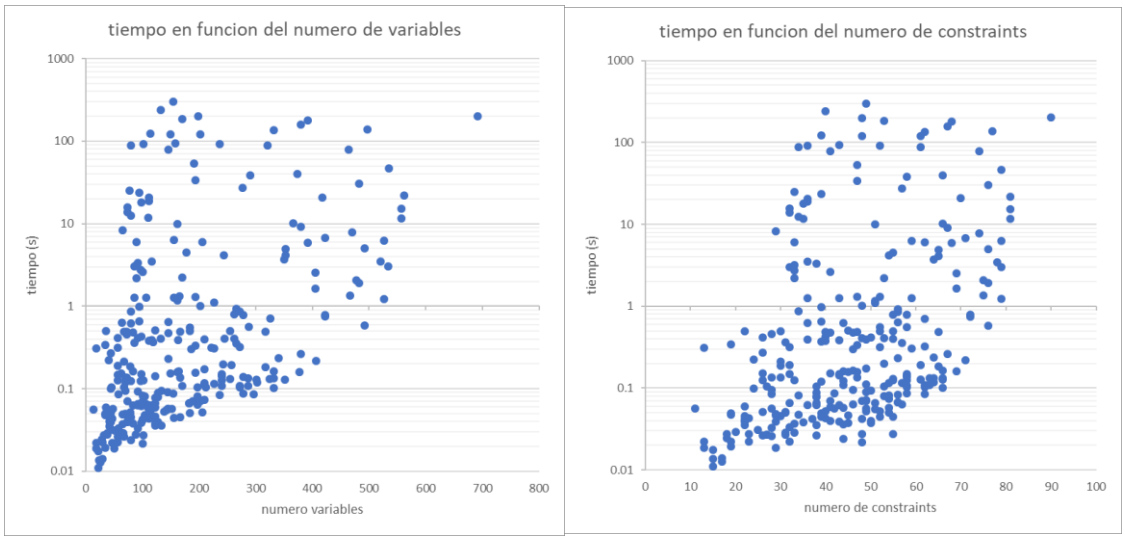
4 – Pruebas realizadas

Para analizar los tiempos se ha utilizado el fichero de entrada $tests\_500\_2\_15\_5\_50.txt$ (generado con el generador), que contiene 500 instancias con valores de n entre 2 y 15, m entre 5 y 50, y v entre 10 y 100000. Estos rangos de valores de los parámetros se seleccionaron tras una serie de pruebas manuales que determinaron que se trataban de valores factibles para su resolución en un límite de tiempo razonable.

En cuanto a la corrección, se ha comprobado de forma no formal, ya que por su naturaleza combinatoria no hay forma rápida de comprobar la optimalidad de una solución, pero en general se han observado valores de la función objetivo muy pequeños en relación a los valores de los bienes v. Por otra parte, si el problema está bien formalizado e implementado (y se permite terminar al algoritmo), la solución será óptima.

A continuación, en las figuras 1 y 2, se muestra el tiempo de resolución de cada instancia en función del número de variables y de restricciones, respectivamente. Solo se tienen en cuenta los que han sido resueltos antes del límite de 5 minutos, que son los que han alcanzado una solución óptima.

Ambos números dependen directamente de n y m. El de variables es $n * (m + 1) + 2$, aproximadamente $n * m$; el de restricciones, $3 * n + m$.



Figuras 1 y 2: Tiempo (s) en función del número de variables y de restricciones, respectivamente

El tiempo debe ser, en general, una función del número o bien de variables o de restricciones (probablemente una combinación de ambas), ya que son los que determinan el tamaño del árbol utilizado por CPLEX.

Más precisamente, permiten prever el tiempo mínimo y máximo de ejecución, ya que las instancias presentan una gran varianza, por la naturaleza del método de resolución del algoritmo que suele solucionarlos (*branch and cut*, similar al *branch and bound* de la práctica anterior, pero combinado con *simplex* o métodos similares).

Esto se observa mejor en la primera gráfica, donde se pueden visualizar que todos los valores se encuentran en la región comprendida entre las dos rectas que parten del origen correspondientes a la función del tiempo mínimo y del máximo.

Aunque no pueda comparar empíricamente estos resultados con los que se obtendrían en un problema de PL clásico (con variables reales), cabe esperar que el último sea mucho más rápido, especialmente para mayores números de parámetros, ya que no tiene la naturaleza combinatoria de los problemas enteros.

Instrucciones para instalación y ejecución

Consultar el README.md adjunto.

Bibliografía

Zwols, Y. *Linear Optimization Solver*, 2014-2015,

https://online-optimizer.appspot.com/?model=ms:vIg8bbKmBqrs7q8uPc6WPQoGTySs8KaR.