Evaluación 3 - Taller de Programación
Problemas combinatoriales resueltos con programación entera mixta
Caso: Knapsack Problem

Prof: Pablo Román

Mayo 27 2024

1 Objetivo

Conocer el problema de programación lineal entera y su resolución mediante el método de Branch and Bound y Simplex. Corresponde a un problema clásico de aplicación en ingeniería industrial en el ámbito de gestión de operaciones. Se implementará en el lenguaje C++, el uso de makefile, la estructuración y buenas prácticas de un código orientado al objeto. Adicionalmente se podrá utilizar los tipos de datos presentes en la librería STL de forma de configurar enfoques mas eficientes de resolución. El problema se simplifica al poder utilizar la rutina simplx del libro Numerical Recipes, la cual ya se encuentra portada a C++. Debe crear las abstracciones necesarias para soportar agregar y eliminar restricciones y poder acelerar el código.

2 El problema de programación lineal entera

Tradicionalmente la programación lineal se define como un problema de optimización en varias variables $X = [x_0, ..., x_{n-1}]$. Dicho problema se caracteriza porque la función a maximizar $Z(X)$ es una suma ponderada de dichas variables $Z = c_0x_0 + ... + c_{n-1}x_{n-1}$. El típico ejemplo corresponde a la ganancia de una empresa donde X son las cantidades de productos producidos y $C = [c_0, ..., c_{n-1}]$ son sus respectivos precios. Adicionalmente el problema esta restringido de forma lineal con desigualdades. Siguiendo con ejemplo anterior, estas desigualdades podrían corresponder restricciones de cantidades de materiales disponibles, presupuestos o tiempos de producción que limitan las cantidades X. En el caso que exista una mezcla de variables enteras y otras que sean continuas se denomina **problema de programación lineal entera mixta**.

Por ser un tipo de problema universal que es común a varias ingenierías se creó una estandarización de dicho problema a lo que se denomina la forma normal. Se tiene entonces la siguiente formulación:

$$\begin{aligned} \max_X Z &= \sum_{k=0}^{n-1} c_k x_k \qquad (1)\\ &\text{s.a.}\\ AX &= b\\ X &\geq 0\\ x_l &\in \mathbb{N}, l \in K\\ x_r &\in \mathbb{R}, r \in R\\ \{0, 1, ..., n-1\} &= K \cup R \end{aligned}$$

En la formulación anterior A es una matriz de $m \times n$, donde n es la cantidad de variables y m es la cantidad de restricciones. El vector b es un dato del problema y es de tamaño m. K corresponde al conjunto de índices que que requiere que las variables x_l sean enteras y R los índices donde las variables son reales.

3 Método Simplex

El método simplex es la base sobre la cual se puede implementar cualquier problema de programación lineal entera mixta (PPLEM o siglas MIP en ingles). Este resuelve el problema 1 sin restricciones enteras, es decir $K = \emptyset$. El

método esta basado en la eliminación Gaussiana en algebra lineal. Para ello se restan filas de las ecuaciones lineales de forma de anular valores en la columna del pivote. El método en detalle se verá en clases y su implementación se encuentra en el libro "Numerical Recipes in C" sección 10.8. En términos geométricos la eliminación Gaussiana va visitando vértices del conjunto factible hasta encontrar el óptimo. Lo que significa que en el peor caso se revisan todos ellos que cuya cantidad es un numero exponencial en n. Aún así en casos prácticos se encuentra el óptimo en una cantidad menor de pasos.

4 Método Branch-and-Bound

El método simplex está diseñando para resolver cuando las variables son continuas. En el caso de tener variables restringidas a números enteros se debe utilizar el método Branch-and-Bound. Para ello se resuelve el sistema relajando las restricciones enteras a continuas mediante el método simplex y se ramifica en la variable mas cercana a un valor entero. Una iteración no optimizada es:

1. Live = {}
2. agregar problema original a Live
3. **For** p : Live
4. resolver p mediante simplex relajando el problema para obtener la solución X
5. si es solución óptima (cota inferior==cota superior) y mejor que cualquier cota en Live → retornar X
6. si es factible y con solucion no optima:
7. encontrar la peor variable x_j y aproximarla a un entero e
8. agregar dos problemas Live con las siguientes restricciones $x_j \geq e+1$ y $x_j \leq e$
9. imprimir no encontró solución
10. retorna vacío

El conjunto Live va a contener los problemas de optimización que hay que revisar. Una forma de optimizar esto es utilizar cota superior e inferior de cada problema. La cota superior Z^{sup} es aquella que se obtiene al relajar el problema (todas las variables son continuas) y resolver con simplex. La cota inferior Z^{inf} es aquella que se obtiene al truncar todas las variables al entero inferior y calcular la función objetivo. Se puede optimizar escogiendo cada vez aquel problema que tenga mayor Z^{sup}. Para ello debería cambiarse la iteración para calcular simplex antes de insertar. El óptimo se llega cuando la cota superior que es la mayor tiene todas sus variables restringidas a enteros en valores enteros es decir $Z^{\text{sup}} == Z^{\text{inf}}$.

5 Problema Knapsack

Se disponen de N itemes de pesos p_i por cada item i y una mochila cuya capacidad máxima es de un peso total P. Cada item i tiene un valor monetario v_i. El asunto es llevar los itemes que maximicen el valor monetario total sin violar la restricción de peso máximo. Este problema se puede formular mediante un problema de programación entera donde $x_i = 1$ si se lleva el item i en la mochila y 0 si no. Esto se puede representar como:

$$\max_X Z = \sum_{k=0}^{N-1} v_k x_k$$

$$\text{s.a.}$$

$$\sum_{k=0}^{N-1} v_k x_k \leq P$$

$$x_l \leq 1$$
$$x_l \geq 0$$
$$x_l \in \mathbb{N} \tag{2}$$

Para acelerar la resolución de este problema mediante heuristicas y/o reducir el tamaño del poliedro factible. Por ejemplo existe una solución mediante un algoritmo greedy que consiste en tomar los itemes de mayor tamaño hasta copar el peso máximo. Dicha solución es una cota inferior que podría agregarse como restricción adicional. Otra heurística que podría utilizar es considerar dividir el conjunto de itemes en partes resolver el problema por cada parte: para construir cotas adicionales y/o resolver exhaustivamente probando todas las separaciones del peso máximo

5.1 Optimizaciones posibles

1. Ordenar el conjunto de nodos por visitar de forma que se tome el que tenga solución con valor mas grande. En el caso que se tenga igual valor entonces se toma la que tenga la cota inferior mas grande. Recordar que la cota inferior se toma truncando la solución a entero y reemplazando en la función objetivo. Esto lleva a ordenar por 2 números.

2. Podar nodos: Si existen nodos cuya cota superior es menor que la cota inferior del nodo que se esta revisando entonces se puede podar. Existen otros criterios de poda.

3. Utilizar solución Greedy para agregar una restricción que disminuya el tamaño del poliedro factible. La solucion greedy corresponde a elegir itemes de manera avara de tal forma que se tome el mayor hasta copar la capacidad de peso del knapsack. Lo que mejor resulta es considerar ordenar los itemes por v_i/p_i y tomar el mayor valor cada vez. La desigualdad que se agrega al sistema es $x_0v_0 + ... + x_{N-1}v_{N-1} \geq V_{greedy}$. (`https://ocw.mit.edu/courses/1-204-computer-algorithms-in-systems-engineering-spring-2010/47c7673f390MIT1_204S10_lec10.pdf`)

4. Cover cut: Se trata de agregar una desigualdad que de la forma por ejemplo $x_{a_1} + ... + x_{a_M} \leq c$ donde los índices a_i es un subconjunto o todo el conjunto $0, ..., N-1$. Si $c < M$ entonces es una desigualdad que realmente corta el conjunto factible en uno menor. Se pueden probar todos los subconjuntos utilizando simplex pero el programa quedaria muy lento. Es interesante probar todos los pares o todo el conjunto menos uno o un subconjunto aleatoreo.

5. Chvatal-gomory cut: Esta optimizacion es mas avanzada pero mejora la busqueda. Se vera en clases.

6 Implementación

Debe implementar una solución lo más eficiente posible para resolver PPLEM. Para testear su algoritmo se debe disponer de un lector de archivos de texto donde se define el problema. Se entregaran una serie de problemas pequeños, medianos y grandes, los cuales sirven para testear y probar eficiencia. Esta vez debe lograr concebir las abstracciones necesarias para lograr implementar el algoritmo de branch-and-bound en forma flexible. Es decir representar un PPLEM que pueda agregar/quitar restricciones en forma flexible y que este pueda ser resuelto mediante el algoritmo simplex. Los objetos deben representar una sola abstracción. El algoritmo de branch-and-bound debe operar sobre dichas abstracciones.

El programa a construir debe estar codificado en C++, implementando el algoritmo para resolver además de las estructuras de datos auxiliares más una heurística que permita acelerar el algoritmo. Dichas estructuras de datos auxiliares pueden ser basadas en los tipos de datos presentes en la librería STL de C++. Debe entregar según la normativa de la tarea #2. Debe cumplir con las normas de orientación al objeto descritas en el documento de formato.

El archivo a recibir por el programa tiene el siguiente formato:

$$N$$
$$v_0v_1...v_{N-1}$$
$$p_0p_1...p_{N-1}$$

Por ejemplo si se tiene 3 itemes y con valores $[1, 2, 3]$ y pesos 0.10.51.2 entonces el archivo es :

```
3
1 2 3
0.1 0.5 1.2
```

Para cargar este archivo se utiliza un menú simple. El programa entregará la solución y el tiempo utilizado.

1. (10%) Informe : contiene una descripción de la estrategia de resolución que se utilizó y el porqué su implementación es eficiente.

2. (30%) Código : se revisa si compila (0 pts si no compila), si se implementa el algoritmo propuesto, si tiene la estructura indicada (clases+test+main+makefile), si se efectúa compilación separada, si esta todo descrito por clases y no hay funciones sueltas salvo la función main, si el makefile opera bien, si el código se entiende (comentarios, variables con nombres descriptivos, identación, sin repeticiones forzadas), si se encuentra 1 test adecuado por clase, si el programa se ejecuta sin problemas y efectúa las operaciones que se espera de la tarea (si no ejecuta en ningún caso 0 pts). **Si tiene funciones fuera de las clases tiene 0 puntos.**

3. (50%) Programa correcto y Eficiencia (0 pts si no funciona.): Se revisa que no existan errores de lógica y/o ejecución, que entregue los resultados correctos en todos los casos (10%). Se revisa que se haya implementado el problema de manera eficiente (40%, 0 si no hay heurística o estructuras de datos eficientes).

4. (10%) Revisión por pares. Cada compañero revisa el código a otro con una nota. Se revisará dicha revisión con otra nota. Si la revisión tiene defectos notables, la nota anterior se reemplaza por otra nota indicada por el profesor. La nota final es la nota propuesta por un par (si es que está correcta) promediada con la evaluación de la revisión que realizó el alumno.

5. (+ 1pto) Bonus al programa que genera la solución en el menor tiempo posible. Si hay varios con la menor cantidad de pasos, entonces se selecciona al programa mas rápido.

7 Entrega

Lunes 17 de Junio 11:55PM entrega final. 1 punto por día de atraso. Archivo entregable es una carpeta con nombre-rut del alumno y comprimida en un archivo con nombre-rut con los archivos requeridos. vía Classroom