Análisis y Diseño de Algoritmos II

Jesús Alexander Aranda Ph.D Robinson Duque, Ph.D
Juan Francisco Díaz, Ph. D

Universidad del Valle

jesus.aranda@correounivalle.edu.co
robinson.duque@correounivalle.edu.co
juanfco.diaz@correounivalle.edu.co
Programa de Ingeniería de Sistemas
Escuela de Ingeniería de Sistemas y Computación

Overview

Programación Lineal (LP)- Conclusiones

- El método simplex proporciona un procedimiento algebraico sistemático para pasar de un punto extremo a otro mientras se mejora el valor de la función objetivo.
- El método fue desarrollado por *George Dantzig* en 1947 y ha sido ampliamente utilizado desde entonces.
- Sin embargo, el algoritmo simplex tiene un mal desempeño en el peor de los casos. Existen familias de problemas de programación lineal para los cuales el método simplex toma una serie de pasos exponenciales en el tamaño del problema.

Programación Lineal (LP)- Conclusiones

- Sólo mucho más tarde, en 1980, se demostró que los programas lineales siempre podían resolverse en tiempo polinomial mediante algo llamado *Algoritmo Elipsoide* propuesto por Khachiyan (pero en la práctica suele ser lento).
- Más adelante, se desarrolló un algoritmo de tiempo polinomial más rápido llamado algoritmo de Karmarkar, que es competitivo con Simplex.
- Con el trabajo de Karmarkar en 1984, se desarrolló el método de *puntos interiores* para programación lineal.

Programación Lineal (LP)- Conclusiones

- En contraste con el algoritmo simplex, que encuentra una solución óptima al atravesar los bordes entre vértices en un conjunto poliédrico, los métodos de puntos interiores se mueven a través del interior de la región factible.
- Hoy en día hay una gran cantidad de algoritmos basados en estos métodos de *punto interior*.

Programación Entera (IP)- Introducción

- Hay muchos problemas donde las variables no son divisibles como fracciones. Algunos ejemplos son el número de operadores que se pueden asignar a los trabajos, el número de aviones que se pueden comprar, el número de plantas en operación, y así sucesivamente.
- Estos problemas forman una clase importante de problemas llamados problemas de *programación entera* (IP).
- En problemas de programación entera, tratar las variables como continuas y redondear la solución óptima al número entero más cercano no siempre es justificable porque puede conllevar a la violación de ciertas restricciones.

Programación Entera (IP)- Introducción

- Sin embargo, también es cierto que en algunos casos, valores cercanos a enteros puedan ser redondeados sin alterar significativamente las características de la solución. Considere el ejemplo del número de barriles de crudo procesado donde la solución resulta en 234566.4 barriles y dedondeamos a 234566.
- Muchas veces, cuando los valores de las variables son pequeños, como el número de aviones en una flota, el redondeo ya no es intuitivo y puede que ni siquiera ofrezca una solución viable.
- En problemas con variables binarias, el redondeo no tiene sentido. La elección entre 0 o 1 es una elección entre dos decisiones completamente diferentes.

Programación Entera (IP)- Introducción

Recordemos el problema sobre la mezcla de alimentos...

Se requiere mezclar dos tipos de alimentos X y Y para alimentar ganado. Cada porción requiere de por lo menos 60 gramos de proteina y por lo menos 30 gramos de grasa. Un paquete de X cuesta \$80 y contiene 15 gramos de de proteina y 10 gramos de grasa; Un paquete de Y cuesta \$50 y contiene 20 gramos de proteina y 5 gramos de grasa.
Modelo final:

$$
\begin{aligned}
\texttt{minimize} \quad & f = 80X + 50Y \\
\texttt{subject to} \quad & 15X + 20Y \geq 60 \\
& 10X + 5Y \geq 30 \\
& X \geq 0 \\
& Y \geq 0
\end{aligned}
$$

Programación Entera (IP)- Introducción

Implementación con variables continuas en MiniZinc:

```
var float: X;
var float: Y;
var float: C;

constraint 15*X + 20*Y >= 60;
constraint 10*X + 5*Y >= 30;
constraint X>=0;
constraint Y>=0;
constraint C=80*X+50*Y;
solve minimize C;
output ["X=", show(X), "\n Y=", show(Y), "\n C=",
    show(C)];
```

Solución: X=2.4, Y=1.2, C=252.0

Programación Entera (IP)- Introducción

Implementación con variables enteras en MiniZinc:

```
var int: X;
var int: Y;
var int: C;

constraint 15*X + 20*Y >= 60;
constraint 10*X + 5*Y >= 30;
constraint X>=0;
constraint Y>=0;
constraint C=80*X+50*Y;
solve minimize C;
output ["X=", show(X), "\n Y=", show(Y), "\n C=",
    show(C)];
```

Solución: X=2, Y=2, C=260

Programación Entera (IP)- Introducción

- ¿De que depende la decisión de utilizar variables enteras y/o continuas?
- ¿Puedo fraccionar paquetes y obtener X=2.4 e Y=1.2?
  - Si la respuesta es si, puedo satisfacer las restricciones a un costo de $252.
  - Si la respuesta es no, debemos quedarnos con la solución entera aunque esto implique a un mayor costo (i.e., $260).
- ¿Que otras implicaciones tiene el uso de variables enteras sobre variables continuas?
  - Variables continuas = Programación Lineal = Algoritmos eficientes (e.g., método de puntos interiores)
  - Variables enteras = Programación entera = ? R/ otras técnicas y ninguna de ellas tiene tiempo polinomial (NP-hard)

Ejercicio

Crear un modelo IP y su implementación en MiniZinc que represente el siguiente problema:

Un fabricante de muebles produce sillas y mesas de jardín. El beneficio de la la venta de cada silla es de $2 y el beneficio de la venta de una mesa es de $3. Cada silla pesa 4 libras y cada mesa pesa 10 libras. Se tiene disponible un suministro de 45 libras en material. El trabajo por silla es de 4 horas y el trabajo por mesa es de 4 horas. Adicionalmente, se tienen disponibles 23 horas de mano de obra. Determine el número de sillas y mesas que se requieren para maximizar las ganancias.

Ejercicios- Fabricante de Muebles

Modelo final:

$$
\begin{aligned}
\texttt{maximize} \quad & 2x_1 + 3x_2 \\
\texttt{subject to} \quad & 4x_1 + 10x_2 \leq 45 \\
& 4x_1 + 4x_2 \leq 23 \\
& x_1, x_2 \geq 0 \\
& x_1, x_2 \ \texttt{son enteros}
\end{aligned}
$$

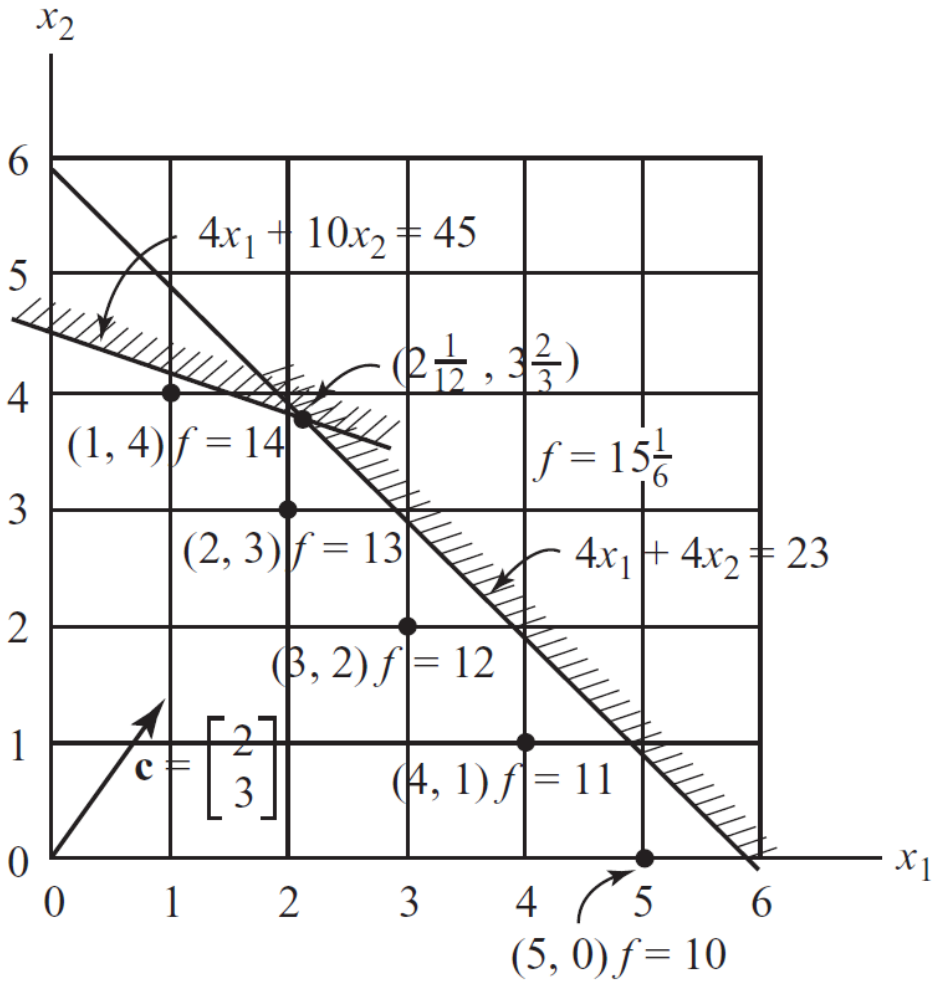
Ejercicios- Fabricante de Muebles

```
var int: X1;
var int: X2;

constraint 4 * X1 + 10 * X2 <= 45;
constraint 4 * X1 + 4 * X2 <= 23;
constraint X1 >= 0;
constraint X2 >= 0;

solve maximize 2 * X1 + 3 * X2;

output ["X1=", show(X1), "\t X2=", show(X2)];
```

Solución: $X1 = 1, X2 = 4$

Ejercicios- Fabricante de Muebles

La representación gráfica del problema se muestra a continuación:



Ejercicios- Fabricante de Muebles

- La solución con Simplex (sin el requerimiento de valores enteros) es $(2\frac{1}{12}, 3\frac{2}{3})$. Cuando se relaja el requerimiento de valores enteros, la ganancia equivale a $15\frac{1}{16}$.
- A continuación se tabulan las soluciones enteras factibles:

Number of chairs	Number of tables	Profit
1	4	14
2	3	13
3	2	12
4	1	11
5	0	10

- Un redondeo intuitivo resultaría en la solución (2, 3) con una ganancia de $13. Mientras que la solución entera resulta en (1,4) con una ganancia de $14.

Branch and Bound- Generalidades

- Branch and Bound (o ramificación y poda) es una técnica para solucionar problemas de programación entera mixta (MIP), propuesta originalmente por Land y Doig en 1960.
- El concepto básico que subyace a la técnica de ramificación y poda es dividir y conquistar.
- Dado que el problema original es difícil de resolver directamente, se divide en subproblemas cada vez más pequeños. Hasta que estos subproblemas puedan ser vencidos.

Branch and Bound- Generalidades

- La división (o ramificación) se realiza mediante la partición del conjunto completo de soluciones factibles en subconjuntos cada vez más pequeños.
- La conquista (o poda) se realiza parcialmente por:
  - dar un límite a la mejor solución en el subconjunto;
  - descartar el subconjunto si el límite indica que no puede contener una solución óptima;

Fin de la Presentación

Ver transparencias con ejemplo sobre branch and bound...

¿Preguntas?