Afi Escuela de Finanzas

Programación Entera

Jorge López Lázaro
jorloplaz@gmail.com

Contenidos

1 Introducción

Introducción

Programación Entera (IP, Integer Programming):

Un modelo de **programación entera** es aquel en el que algunas o todas las variables deben tomar valores enteros.

Introducción

Programación Entera (IP, Integer Programming):

Un modelo de **programación entera** es aquel en el que algunas o todas las variables deben tomar valores enteros.

- **Programación entera mixta**: algunas de las variables toman valores enteros.
- **Programación entera pura**: todas las variables toman valores enteros.
- **Programación binaria:** $\boldsymbol{x} \in \{\mathbf{0}, \mathbf{1}\}$ variables de asignación, lógicas.

Introducción

Programación Entera (IP, Integer Programming):

Un modelo de **programación entera** es aquel en el que algunas o todas las variables deben tomar valores enteros.

- **Programación entera mixta**: algunas de las variables toman valores enteros.
- **Programación entera pura**: todas las variables toman valores enteros.
- **Programación binaria:** $\boldsymbol{x} \in \{\mathbf{0}, \mathbf{1}\}$ variables de asignación, lógicas.

Estudiaremos los problemas de programación lineal entera, es decir, problemas de programación lineal con alguna variable entera.

Introducción

- Son más difíciles de resolver que los problemas LP, porque se pierde la convexidad.
- El coste computacional para encontrar la solución de estos problemas es alto. Algunos problemas con pocas decenas de variables son casi imposibles de resolver en un tiempo razonable.
- La programación entera mixta (MIP) ha mejorado bastante en las últimas décadas.
- Un problema de MIP que hace 20 años tardaba 7 años en resolverse, actualmente, en el mismo ordenador, se resuelve en menos de un segundo.

Introducción

Uno de los problemas más conocidos de programación entera es TSP (***Traveling Salesman Problem***), el problema del viajante.

Introducción

Uno de los problemas más conocidos de programación entera es TSP (***Traveling Salesman Problem***), el problema del viajante.

Un vendedor ambulante tiene que visitar varios clientes y volver a casa para cenar.

Dada una lista de ciudades y las distancias entre cada par de ellas,

¿Cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?

2 | Formulación de programación entera

Formulación de programación entera

Formulación general

$$\min / \max_{x_1,\ldots,x_n} \quad z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \quad \sum_{j=1}^{n} A_{ij} x_j = b_i, \quad i = 1, \ldots, m_e$$

$$\sum_{j=1}^{n} A_{ij} x_j \geq b_i, \quad i = 1, \ldots, m_g$$

$$\sum_{j=1}^{n} A_{ij} x_j \leq b_i, \quad i = 1, \ldots, m_l$$

$$x_j \text{ entero para } \quad j \in J \subseteq \{1, \ldots, n\},$$

donde $m = m_e + m_g + m_l$ es el número total de restricciones, y $c_j, A_{ij}\ y\ b_j, para\ todo\ i = 1, \ldots, m, y\ todo\ j = 1, \ldots, n$, son parámetros del problema.

Formulación de programación entera

Variables binarias (0-1)

- Útiles para representar problemas de decisiones de tipo “sí/no”.
- $x_i = 1$ si seleccionamos la alternativa *i,* o $x_i = 0$ en caso contrario.
- Aparece en muchos modelos que abordan decisiones estratégicas de alto costo, por ejemplo, decisión de inversión de capital.
- Flexibilidad en el modelo. Por ejemplo, si de todas las actividades podemos seleccionar como mucho *n* de ellas:

$$\sum_i x_i \leq n$$

Formulación de programación entera

Restricciones lógicas → *k-Fold* restricciones alternativas

Se pueden usar variables binarias para forzar asociaciones entre variables. Aparecen cuando el valor de una variable binaria impone condiciones sobre otras.

- Si el proyecto y no se lleva a cabo, el proyecto x tampoco:

$$y = 0 \Rightarrow x = 0 \; equivalente \; a \; x \leq y, \; con \; x, y \in \{0,1\}$$

- Si el proyecto y no se lleva a cabo, el proyecto x se debe llevar a cabo:

$$y = 0 \Rightarrow x = 1 \; equivalente \; a \; x \geq 1 \; - \; y, \; con \; x, y \in \{0,1\}$$

- Si el proyecto y se lleva a cabo, el proyecto x no se puede llevar a cabo:

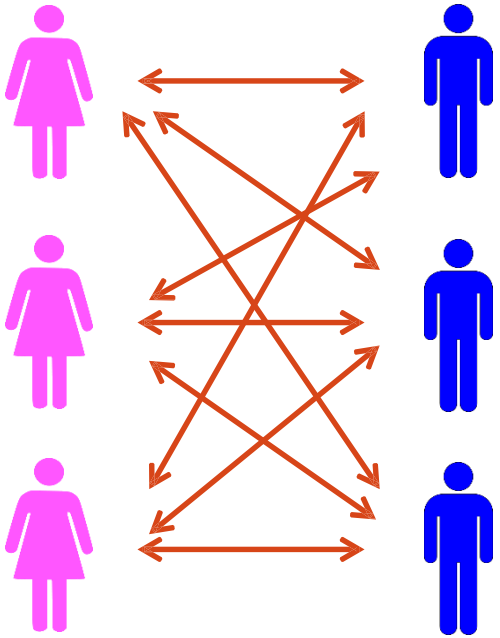
$$y = 1 \Rightarrow x = 0 \; equivalente \; a \; x \leq \; 1 \; - \; y, \; con \; x, y \in \{0,1\}$$

- Si el proyecto y se lleva a cabo, el proyecto x también:

$$y = 1 \Rightarrow x = 1 \; equivalente \; a \; x \geq \; y, \; con \; x, y \in \{0,1\}$$

3 Ejemplos

Ejemplo 1: Web de citas



- Cada usuario debe rellenar un cuestionario largo, que define las “dimensiones de la personalidad”.
- Basado en esto, la página web estima los score de compatibilidad asociados a cada posible pareja.
- ¿Cómo podemos emparejar a las personas para conseguir la **compatibilidad global máxima**?

Ejemplo 1: Web de citas

Sea $i = 1,2,3$ el índice que representa a las mujeres y $j = 1,2,3$ el índice que representa a los hombres. Sea w_{ij} el score de compatibilidad entre la mujer *i* y el hombre *j.*

Variables de decisión

$$x_{ij} = \begin{cases} 1, & emparejamos\ el\ usuario\ i\ con\ el\ usuario\ j \\ 0, & no\ emparejamos\ el\ usuario\ i\ con\ el\ usuario\ j \end{cases}$$

Objetivo: maximizar la compatibilidad global

$$w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33}$$

Restricciones

Ejemplo 1: Web de citas

Restricciones

Unir cada usuario *i* con un usuario *j* y viceversa

$$x_{11} + x_{12} + x_{13} = 1$$
$$x_{21} + x_{22} + x_{23} = 1$$
$$x_{31} + x_{32} + x_{33} = 1$$
$$x_{11} + x_{21} + x_{31} = 1$$
$$x_{12} + x_{22} + x_{32} = 1$$
$$x_{13} + x_{23} + x_{33} = 1$$

Variables binarias

$$x_{ij} \in \{0,1\}, \quad i = 1,2,3 \quad j = 1,2,3$$

Ejemplo 1: Web de citas

Formulación general

$$\max_{x_{ij}} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_{ij}$$

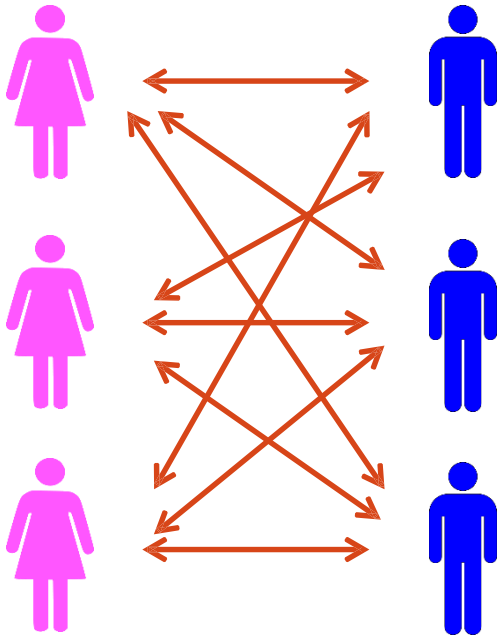
$$\text{s.t.:}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i$$

$$x_{ij} \in \{0,1\} \quad \forall ij$$

Ejemplo 1: Web de citas



¿Cuál es la solución óptima si las compatibilidades son…?

w_{ij}	$i=1$	$i=2$	$i=3$
$j=1$	1	3	5
$j=2$	4	2	2
$j=3$	1	5	3

- Solución con Python.
- ¿Cómo podemos modificar la formulación si hay más hombres que mujeres?

Ejemplo 2: Selección de un proyecto

- Una compañía está evaluando 5 proyectos para desarrollar en los próximos 3 años. Estos proyectos requieren una inversión durante dichos años.
- Hay un máximo de presupuesto de 25 millones de euros por año.
- ¿Qué proyectos deben elegir para **maximizar** los beneficios totales?
- El beneficio esperado de cada proyecto (en millones de euros) y la inversión que se necesita hacer para cada proyecto y año (millones de euros) son:

Proyecto	INVERSIÓN (millones euros/año) 1	2	3	Beneficio
1	5	1	8	20
2	4	7	10	40
3	3	9	2	20
4	7	4	1	15
5	8	6	10	30

Ejemplo 2: Selección de un proyecto

Variables de decisión

$$x_i = \begin{cases} 1, & si\ se\ selecciona\ el\ proyecto\ i \\ 0, & si\ no\ se\ selecciona\ el\ proycto\ i \end{cases}$$

Objetivo: maximizar los beneficios totales esperados.

$$20x_1 + 40x_2 + 20x_3 + 15x_4 + 30x_5$$

Restricciones:
Presupuesto anual del proyecto

$$5x_1 + 4x_2 + 3x_3 + 7x_4 + 8x_5 \leq 25 \quad \text{(presupuesto del año 1)}$$
$$x_1 + 7x_2 + 9x_3 + 4x_4 + 6x_5 \leq 25 \quad \text{(presupuesto del año 2)}$$
$$8x_1 + 10x_2 + 2x_3 + x_4 + 10x_5 \leq 25 \quad \text{(presupuesto del año 3)}$$

Variables binarias

$$x_i \in \{0,1\}, i = 1,2,3,4,5$$

Ejemplo 2: Selección de un proyecto

- **Solución óptima:** Python (**ejercicio**)

Ejemplo 2: Selección de un proyecto

- **Solución óptima:** Python (**ejercicio**)
- ¿Qué restricción debemos añadir si necesitamos imponer que no se seleccionen más de tres proyectos?

Ejemplo 2: Selección de un proyecto

- **Solución óptima:** Python (**ejercicio**)

- ¿Qué restricción debemos añadir si necesitamos imponer que no se seleccionen más de tres proyectos?

$$x_1 + x_2 + x_3 + x_4 + x_5 \leq 3$$

Ejemplo 3: Problema del viajante

Un vendedor ambulante tiene que visitar varios clientes y volver a casa para cenar.

Dada una lista de ciudades y las distancias entre cada par de ellas,

¿Cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?

Ejemplo 3: Problema del viajante

Ejemplo 3: Problema del viajante

Ejemplo 3: Problema del viajante

Variables de decisión

Para cada par de nodos $i, j \in N$ definimos

$$x_{ij} = \begin{cases} 1, & si\ se\ visita\ el\ nodo\ j\ justo\ después\ que\ el\ nodo\ i \\ 0, & en\ caso\ contrario \end{cases}$$

Objetivo

$$minimize \sum_{i \neq j} c_{ij} x_{ij}$$

Restricciones

Se debe llegar a cada nodo desde exactamente un nodo:

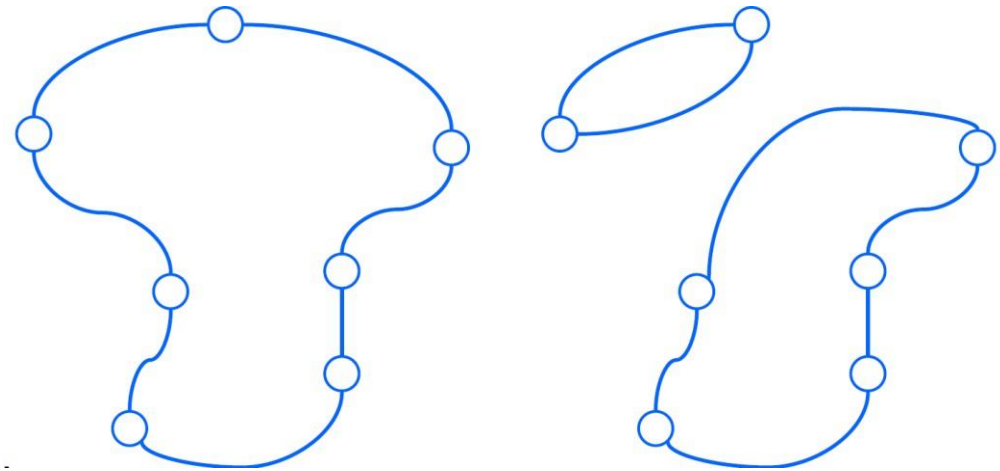
$$\sum_{i} x_{ij} = 1$$

Desde cada nodo se debe salir sólo a un único nodo:

$$\sum_{j} x_{ij} = 1$$

Ejemplo 3: Problema del viajante

- Con este modelo podemos encontrar soluciones de este tipo:



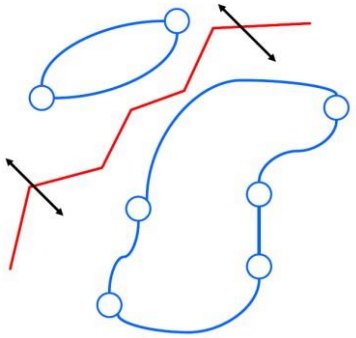
- ¿Qué podemos hacer para evitar los recorridos cerrados (ciclos)?

Ejemplo 3: Problema del viajante

Algunas opciones:

- Si obtenemos ciclos, podemos “romperlos” incluyendo más restricciones:

$$Para\ cada\ ciclo\ S \subseteq N\ con\ |S| \leq \frac{n}{2}: \sum_{a \in \delta(S)} x_a \geq 1,$$
$$donde\ \delta(S)\ es\ el\ conjunto\ de\ ejes\ con\ un\ extremo\ en\ S\ y\ otro\ fuera\ de\ S.$$



Esto obliga a unir los ciclos

- Añadir el conjunto MTZ (Miller-Tucker-Zemlin) a las restricciones:

$$u_1 = 1$$
$$2 \leq u_i \leq n, \forall i \neq 1$$
$$u_i - u_j + 1 \leq (n-1)\left(1 - x_{ij}\right) \forall i \neq 1, \forall j \neq 1$$
$$donde\ u_i, con\ i = 1, \dots, n\ son\ variables\ extras.$$

Esto obliga a incluir el nodo 1 en todos los ciclos

http://materias.fi.uba.ar/7114/Docs/ApunteViajante.pdf

Ejemplo 3: Problema del viajante

¿Cuántas restricciones tenemos?

- 5 nodos: 16 restricciones
- 10 nodos: 638 restricciones
- 25 nodos: 16777216 restricciones

No es viable aplicar el modelo de *Branch and bound* al modelo completo ➔ Métodos heurísticos

4 | Algoritmos

Algoritmos para encontrar la solución

- Herramientas básicas:

  Resolver problemas lineales relajados, PR (sin restricciones de números enteros) → La solución quizá no es factible.

  Si P.R. es imposible => P.E es imposible

  Si P.R. es ilimitado => P.E. es ilimitado o imposible

  Si P.R. tiene S.O => P.E tiene S.O o es imposible

Algoritmos para encontrar la solución

- Estrategia *Naive*:

  Redondeo → Redondear los números de la solución obtenida del problema lineal relajado.

  - Hay diferentes alternativas para redondear dichos números (exponencial)
  - No hay garantía de que se llegue a la solución óptima.
  - Incluso no hay garantía de que se llegue a una solución factible.

Los **métodos exactos** se usan para resolver problemas **de tamaño moderado.**

Para **problemas grandes** tardan mucho, por lo que se usan **métodos heurísticos.**

Algoritmos para encontrar la solución

Algoritmos eficientes:

- *Branch and bound* (ramificación y acotamiento) → Divide el problema introduciendo condiciones simples sobre los valores de las variables.
- *Cutting planes* (planos de corte)→ Introduce restricciones para forzar a la solución entera que sea un vértice del problema modificado.
- *Branch and cut* (ramificación y poda) → Una combinación de los métodos anteriores.

Branch and bound es el que más se usa, por lo que vamos a verlo en detalle.

Ramificación y acotación (*Branch and bound*)

Paso 1: resolver el problema relajado asociado al problema entero.

- Si la solución óptima es entera, esa solución es óptima también para el problema entero. Fin
- En otro caso, fijar una cota z_i inferior (en caso de maximizar) para el valor óptimo del problema entero. Si no se conoce ninguna solución candidata para el problema entero, $z_i = -\infty$

Paso 2: **Ramificación.** Seleccionar un problema no terminal (aquel cuya solución no es entera). Elegir una variable x_j que tome un valor no entero en la solución actual (sabemos que al final debe ser entero). Crear dos nuevos problemas añadiendo al problema las restricciones $x_j \leq [x_j]$ y $x_j \geq [x_j] + 1$, donde $[x_j]$ indica la parte entera de x_j.

Paso 3: **Acotación.** Resolver cada uno de los dos problemas creados en el paso anterior.

Ramificación y acotación (*Branch and bound*)

Paso 4: **Problemas terminales.** Analizar los problemas que puedan contener la solución óptima y considerar terminales los que cumplen una de las siguientes condiciones:

1) El problema es no factible.
2) $z_s \leq z_i$
3) $z_S > z_i$ y la solución es entera. Se actualiza la cota haciendo $z_i = z_s$ y esta solución entera es la solución candidata.

Si todos los problemas son terminales, la solución candidata es la solución óptima. Si no hay solución candidata, el problema entero es no factible.

Si hay problemas no terminales, volver al Paso 2 para continuar con el proceso de ramificación.

Branch and bound (ejemplo)

$$
\begin{aligned}
\max \quad & Z = -x_1 + 4x_2 \\
s.t. \quad & \\
& -10x_1 + 20x_2 \geq 22 \\
& 5x_1 + 10x_2 \leq 49 \\
& x_1 \leq 5 \\
& x_1, x_2 \in \mathbb{Z}^+
\end{aligned}
$$

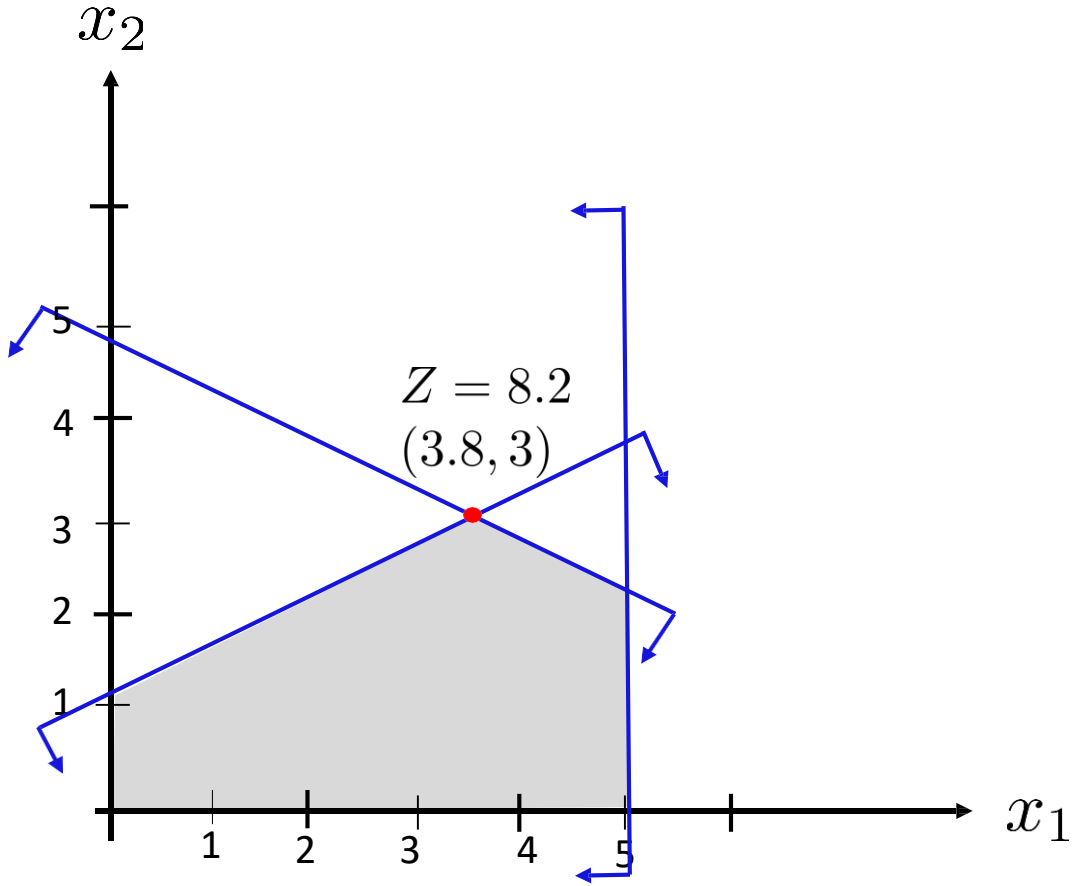
Relajación:

$$
\begin{aligned}
\max \quad & Z = -x_1 + 4x_2 \\
s.t. \quad & \\
& -10x_1 + 20x_2 \geq 22 \\
& 5x_1 + 10x_2 \leq 49 \\
& 0 \leq x_1 \leq 5 \\
& x_2 \geq 0
\end{aligned}
$$

Branch and bound (ejemplo)

Relajación

$$\begin{aligned} \max \quad & Z = -x_1 + 4x_2 \\ s.t. \quad & \\ & -10x_1 + 20x_2 \geq 22 \\ & 5x_1 + 10x_2 \leq 49 \\ & 0 \leq x_1 \leq 5 \\ & x_2 \geq 0 \end{aligned}$$



Branch and bound (ejemplo)

Relajación

$$\begin{aligned} \max \quad & Z = -x_1 + 4x_2 \\ s.t. \quad & \\ & -10x_1 + 20x_2 \geq 22 \\ & 5x_1 + 10x_2 \leq 49 \\ & 0 \leq x_1 \leq 5 \\ & x_2 \geq 0 \end{aligned}$$

$Z = 8.2$
$(3.8, 3)$

P1

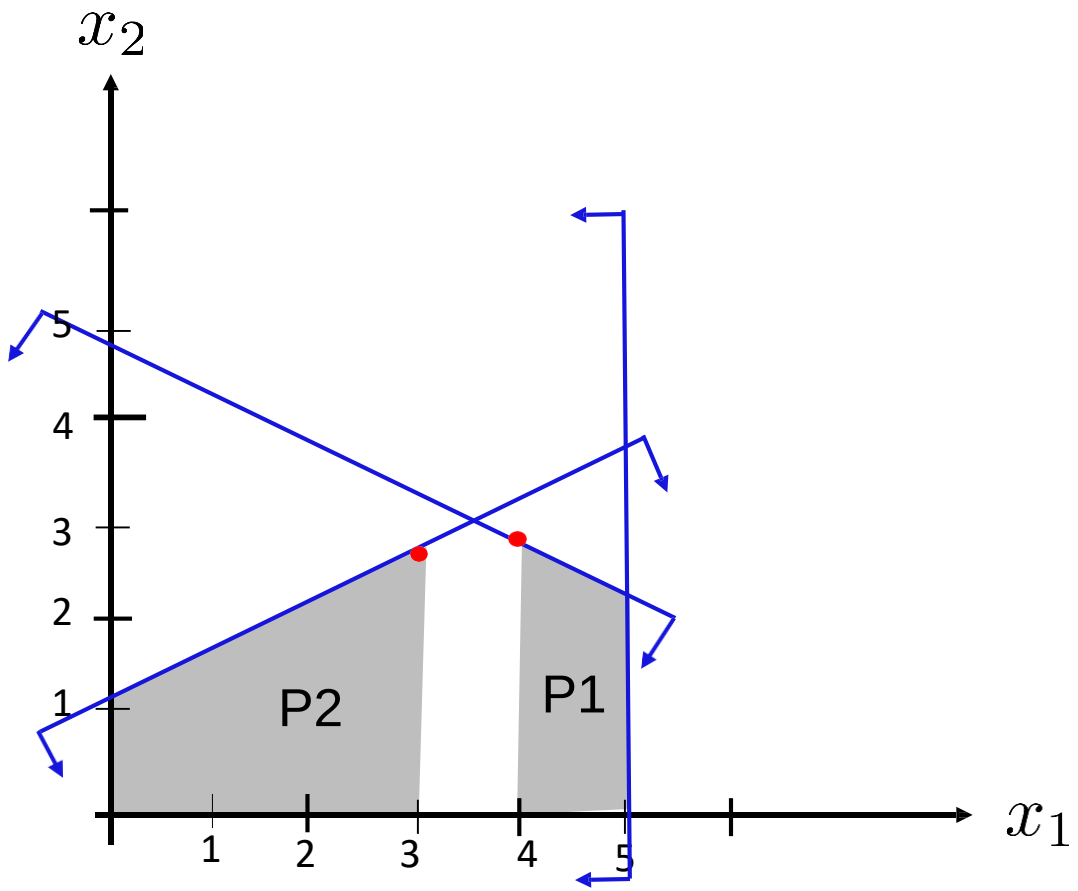
$$\begin{aligned} \max \quad & Z = -x_1 + 4x_2 \\ s.t. \quad & \\ & -10x_1 + 20x_2 \geq 22 \\ & 5x_1 + 10x_2 \leq 49 \\ & 0 \leq x_1 \leq 5 \\ & x_1 \geq 4 \\ & x_2 \geq 0 \end{aligned}$$

7.6
$(4, 2.9)$

P2

$$\begin{aligned} \max \quad & Z = -x_1 + 4x_2 \\ s.t. \quad & \\ & -10x_1 + 20x_2 \geq 22 \\ & 5x_1 + 10x_2 \leq 49 \\ & 0 \leq x_1 \leq 5 \\ & x_1 \leq 3 \\ & x_2 \geq 0 \end{aligned}$$

$Z = 7.4$
$(3, 2.6)$



Branch and bound (ejemplo)

P1

$$\begin{aligned} \max \quad & Z = -x_1 + 4x_2 \\ s.t. \quad & \\ & -10x_1 + 20x_2 \geq 22 \\ & 5x_1 + 10x_2 \leq 49 \\ & 0 \leq x_1 \leq 5 \\ & x_1 \geq 4 \\ & x_2 \geq 0 \end{aligned}$$

7.6
$(4, 2.9)$

P11

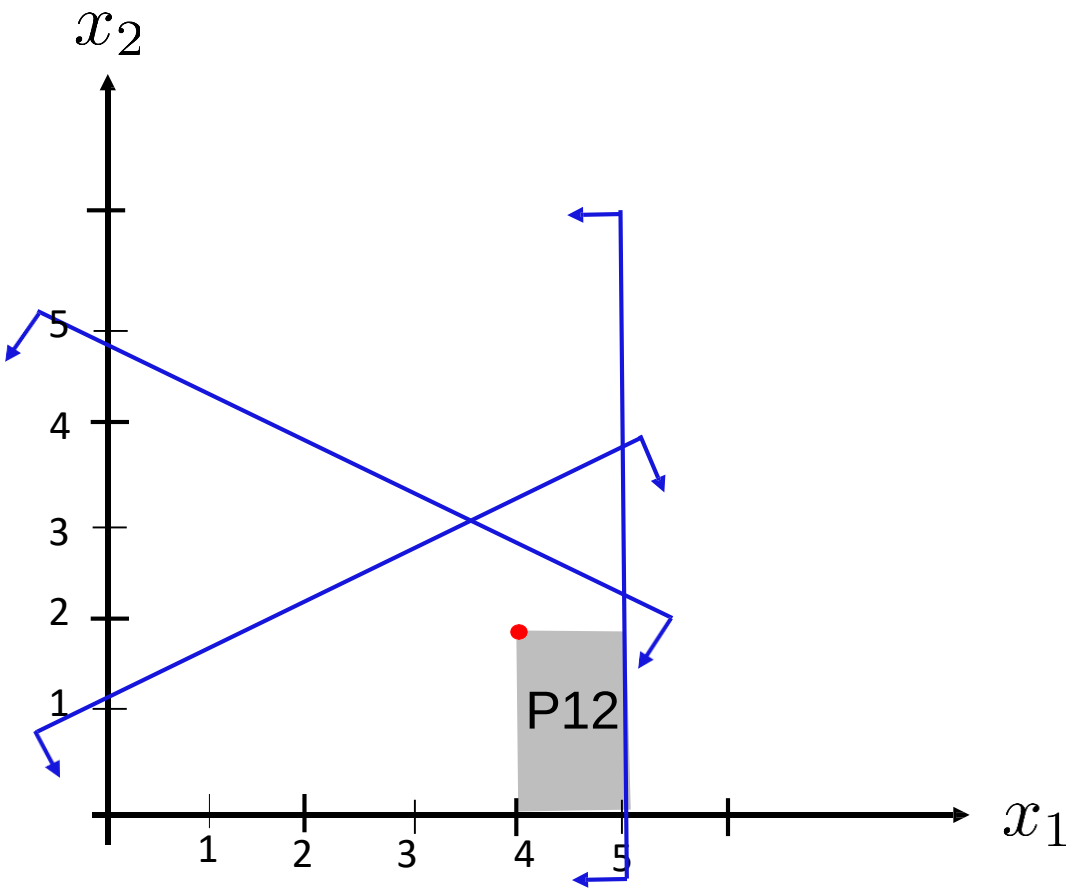
$$\begin{aligned} \max \quad & Z = -x_1 + 4x_2 \\ s.t. \quad & \\ & -10x_1 + 20x_2 \geq 22 \\ & 5x_1 + 10x_2 \leq 49 \\ & 0 \leq x_1 \leq 5 \\ & x_1 \geq 4 \\ & x_2 \geq 3 \\ & x_2 \geq 0 \end{aligned}$$

Inviable

P12

$$\begin{aligned} \max \quad & Z = -x_1 + 4x_2 \\ s.t. \quad & \\ & -10x_1 + 20x_2 \geq 22 \\ & 5x_1 + 10x_2 \leq 49 \\ & 0 \leq x_1 \leq 5 \\ & x_1 \geq 4 \\ & 0 \leq x_2 \leq 2 \end{aligned}$$

$Z = 4$
$(4, 2)$



Branch and bound (ejemplo)

P2

$$\max \quad Z = -x_1 + 4x_2$$
$$s.t.$$
$$-10x_1 + 20x_2 \geq 22$$
$$5x_1 + 10x_2 \leq 49$$
$$0 \leq x_1 \leq 5$$
$$x_1 \leq 3$$
$$x_2 \geq 0$$

$Z = 7.4$
$(3, 2.6)$

P21

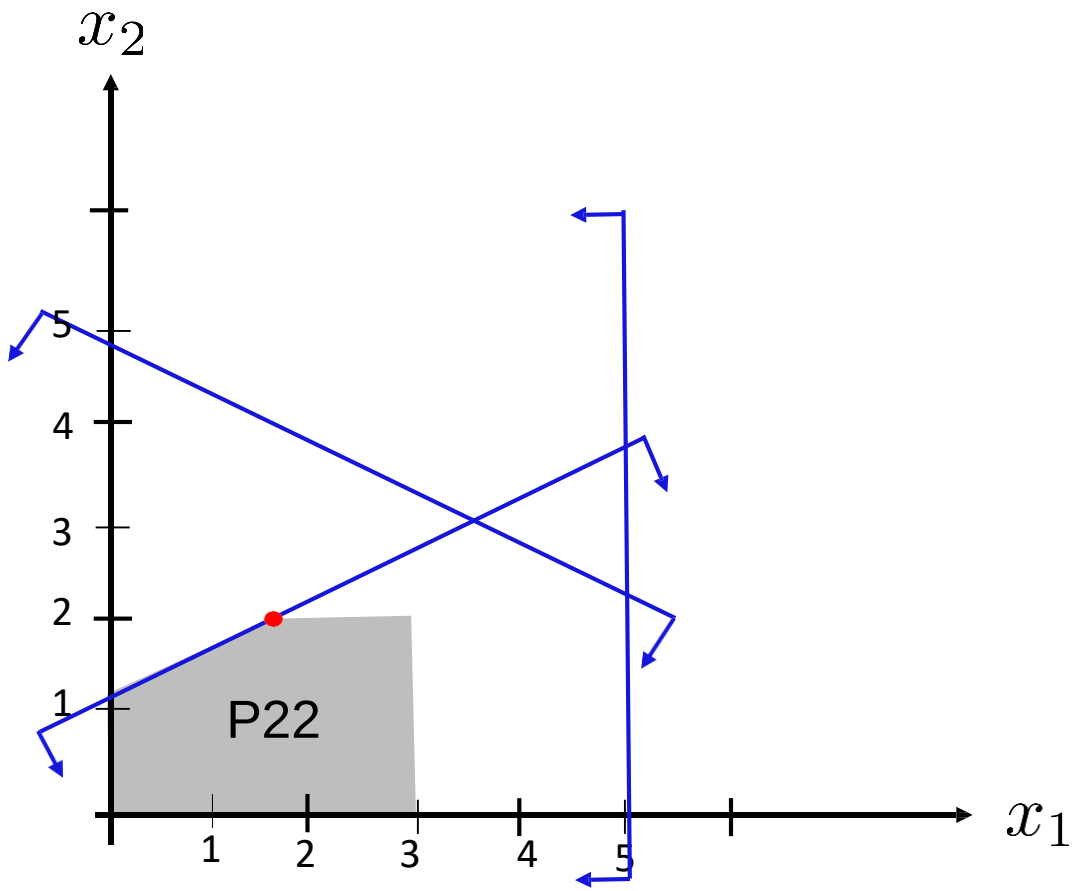
$$\max \quad Z = -x_1 + 4x_2$$
$$s.t.$$
$$-10x_1 + 20x_2 \geq 22$$
$$5x_1 + 10x_2 \leq 49$$
$$0 \leq x_1 \leq 5$$
$$x_1 \leq 3$$
$$x_2 \geq 0$$
$$x_2 \geq 3$$

Inviable

P22

$$\max \quad Z = -x_1 + 4x_2$$
$$s.t.$$
$$-10x_1 + 20x_2 \geq 22$$
$$5x_1 + 10x_2 \leq 49$$
$$0 \leq x_1 \leq 5$$
$$x_1 \leq 3$$
$$x_2 \geq 0$$
$$x_2 \leq 2$$

$Z = 6.2$
$(1.8, 2)$



Branch and bound (ejemplo)

P22

$$\max \quad Z = -x_1 + 4x_2$$
$$s.t.$$
$$-10x_1 + 20x_2 \geq 22$$
$$5x_1 + 10x_2 \leq 49$$
$$0 \leq x_1 \leq 5$$
$$x_1 \leq 3$$
$$x_2 \geq 0$$
$$x_2 \leq 2$$

$Z = 6.2$
$(1.8, 2)$

P221

$$\max \quad Z = -x_1 + 4x_2$$
$$s.t.$$
$$-10x_1 + 20x_2 \geq 22$$
$$5x_1 + 10x_2 \leq 49$$
$$0 \leq x_1 \leq 5$$
$$2 \leq x_1 \leq 3$$
$$x_2 \geq 0$$
$$x_2 \leq 2$$

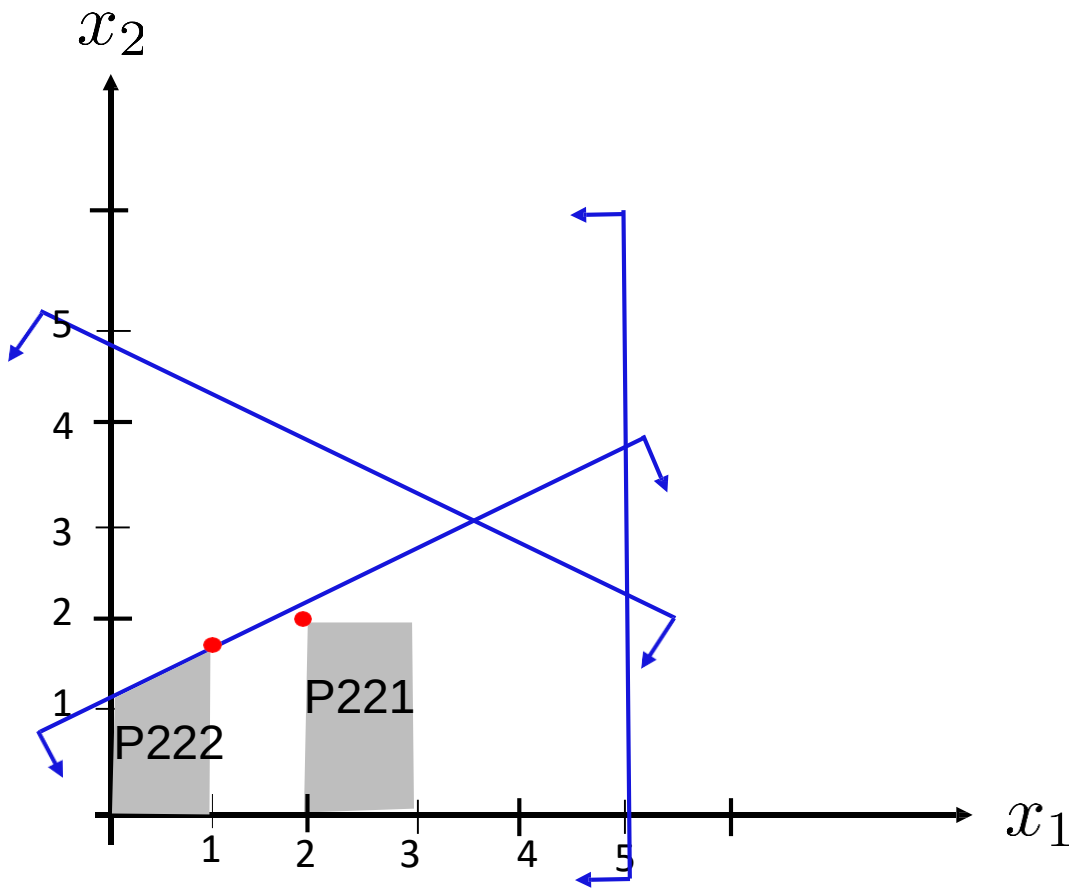
$Z = 6$
$(2, 2)$

$\geq$

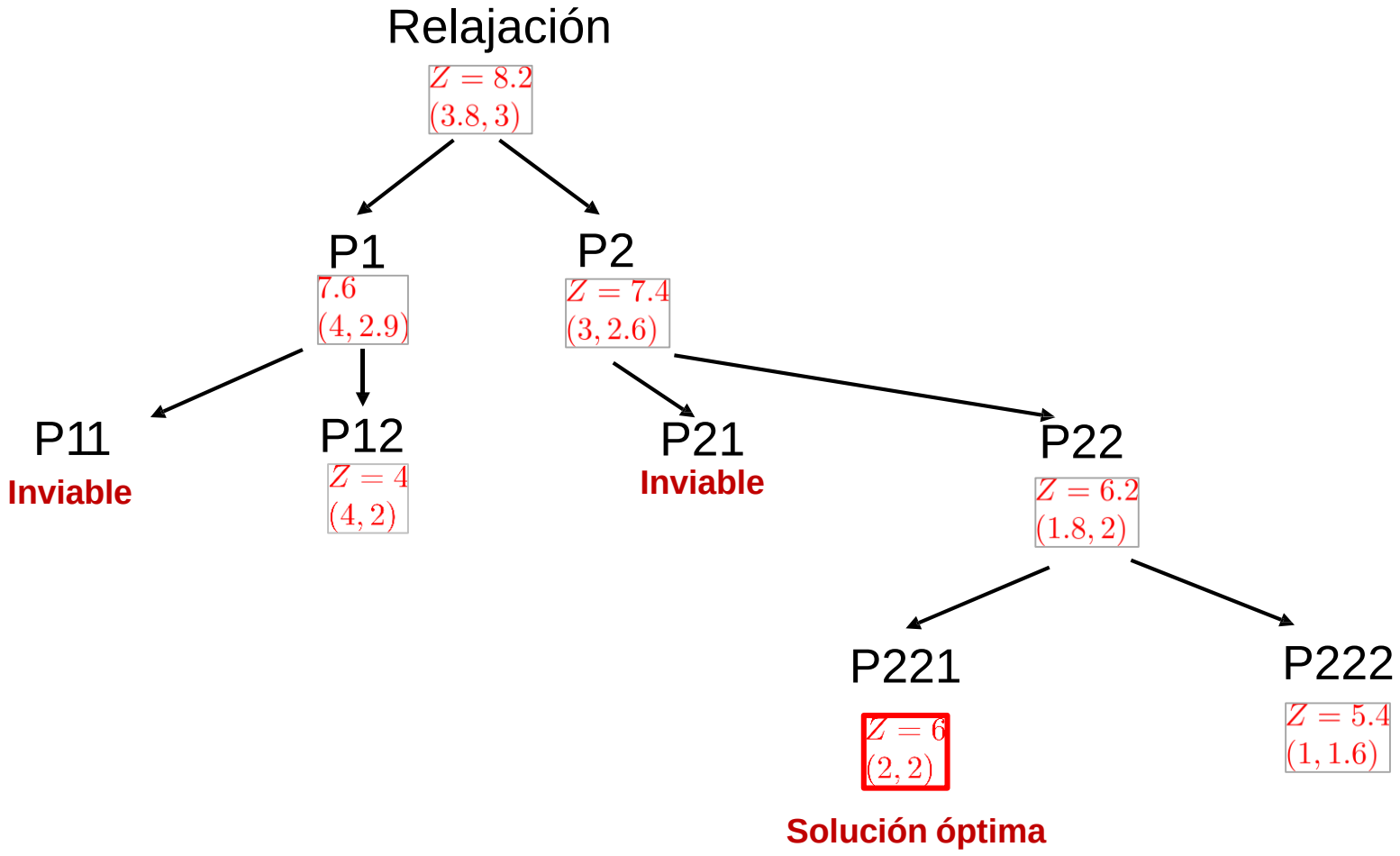
P222

$$\max \quad Z = -x_1 + 4x_2$$
$$s.t.$$
$$-10x_1 + 20x_2 \geq 22$$
$$5x_1 + 10x_2 \leq 49$$
$$0 \leq x_1 \leq 5$$
$$x_1 \leq 3$$
$$x_1 \leq 1$$
$$x_2 \geq 0$$
$$x_2 \leq 2$$

$Z = 5.4$
$(1, 1.6)$



Branch and bound (ejemplo)



Ramificación y acotación: Ejercicio

$$
\begin{aligned}
Min \quad & -5x_1 - 4x_2 \\
s.a \quad & x_1 + x_2 \leq 5 \\
& 10x_1 - 6x_2 \leq 45 \\
& x_1, x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{aligned}
$$

Ramificación y acotación: Ejercicio

$$\begin{aligned} Min \quad & -5x_1 - 4x_2 \\ s.a \quad & x_1 + x_2 \leq 5 \\ & 10x_1 + 6x_2 \leq 45 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned}$$

Problema relajado

$$\begin{aligned} Min \quad & -5x_1 - 4x_2 \\ s.a \quad & x_1 + x_2 \leq 5 \\ & 10x_1 + 6x_2 \leq 45 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Solución problema relajado

$$X^0 = (3.75, 1.25)$$
$$z^0 = -23{,}75$$

Ejercicio. Resuelve el problema entero mediante el algoritmo de ramificación y acotación.

Nota: Puedes resolver cada subproblema con Python, pero debes escribir el planteamiento de cada uno.

Métodos heurísticos para encontrar la solución

Los métodos nombrados anteriormente no resuelven cualquier problema de programación entera. La mayoría de los métodos que se usan son **heurísticos: métodos constructivos y métodos de mejora.**

Métodos heurísticos para encontrar la solución

Los métodos nombrados anteriormente no resuelven cualquier problema de programación entera. La mayoría de los métodos que se usan son **heurísticos: métodos constructivos y métodos de mejora.**

Un método heurístico proporciona una buena solución del problema, no necesariamente óptima.

Métodos heurísticos para encontrar la solución

Los métodos nombrados anteriormente no resuelven cualquier problema de programación entera. La mayoría de los métodos que se usan son **heurísticos: métodos constructivos y métodos de mejora.**

Un método heurístico proporciona una buena solución del problema, no necesariamente óptima.

Un buen algoritmo heurístico debe ser: **eficiente, bueno y robusto.**

Eficiente: esfuerzo computacional realista.
Bueno: la solución en promedio está cerca del óptimo.
Robusto: la probabilidad de obtener una mala solución es baja.

Métodos heurísticos para encontrar la solución

Heurísticas de construcción

Un procedimiento de construcción se basa en **construir paso a paso una solución del problema**. Usualmente son métodos deterministas y suelen estar basados en la mejor elección en cada iteración.

Estos métodos han sido muy utilizados en problemas clásicos, como el del viajante.

Métodos heurísticos para encontrar la solución

Heurísticas de construcción

Miopía de las heurísticas constructivas (I)

Algunas heurísticas constructivas se denominan **voraces** (“greedy”). Este calificativo se debe a que avanzan sin posibilidad de deshacer las decisiones tomadas. Las decisiones tomadas en cada etapa, aunque buenas según el criterio de selección, perjudican la solución final.

También se les califica de “**miopes**”, ya que sólo ven lo más cercano, no teniendo una visión global.

Métodos heurísticos para encontrar la solución

Heurísticas de construcción

Miopía de las heurísticas constructivas (II)

Hay dos formas de solucionar este problema:

- Dotar de **profundidad** a la regla de decisión. Esto es, que en cada etapa de decisión se tenga en cuenta, en la medida de lo posible, cómo afectará en las siguientes etapas la decisión actual.

- Dotar a las heurísticas de técnicas de ***backtracking***. Es decir, dotar de capacidad a las heurísticas de volver atrás en las decisiones (y tomar nuevas decisiones) si la situación no es la deseable.

Métodos heurísticos para encontrar la solución

Heurísticas de mejora

Un **procedimiento de mejora** parte de una solución inicial y escoge una nueva solución en su **vecindario** de acuerdo a algún criterio de selección.

Métodos heurísticos para encontrar la solución

Heurísticas de mejora

Un **procedimiento de mejora** parte de una solución inicial y escoge una nueva solución en su **vecindario** de acuerdo a algún criterio de selección.

Uno de los criterios más simples es tomar una solución con menor valor de la función objetivo que la actual. Este criterio, conocido como “greedy”, permite ir mejorando la solución actual mientras se pueda. El algoritmo se detiene cuando la solución no puede ser mejorada.

Son métodos muy rápidos y, pese a su miopía, proporcionan soluciones que, en promedio, están relativamente cerca del óptimo global.

Métodos heurísticos para encontrar la solución

Heurísticas de mejora

Vecindario

El vecindario de una solución es el conjunto de soluciones que pueden ser **alcanzados** desde dicha solución con una **operación simple**.

Cualquier solución que requiera dos operaciones **no** es del vecindario.

La mejor solución en un vecindario es un óptimo con respecto a su vecindario.

Solvers

CPLEX, Gurobi, GLPK, LAMPS, SBB, XPRESS…

Muchos de ellos usan *Branch and Cut.*

5 Referencias

Referencias

- Bradley, S., Hax, A., & Magnanti, T. (1977). Applied mathematical programming.
- Castillo, E., Conejo A. J., Pedregal P., Garcia R., Alguacil, N.: Building and Solving Mathematical Programming Models in Engineering and Science. John Wiley & Sons, New York (2001)

En la web

http://www.est.uc3m.es/esp/nueva_docencia/comp_col_leg/ing_info/io/doc_generica/archivos/pe.pdf

Afi Escuela de Finanzas