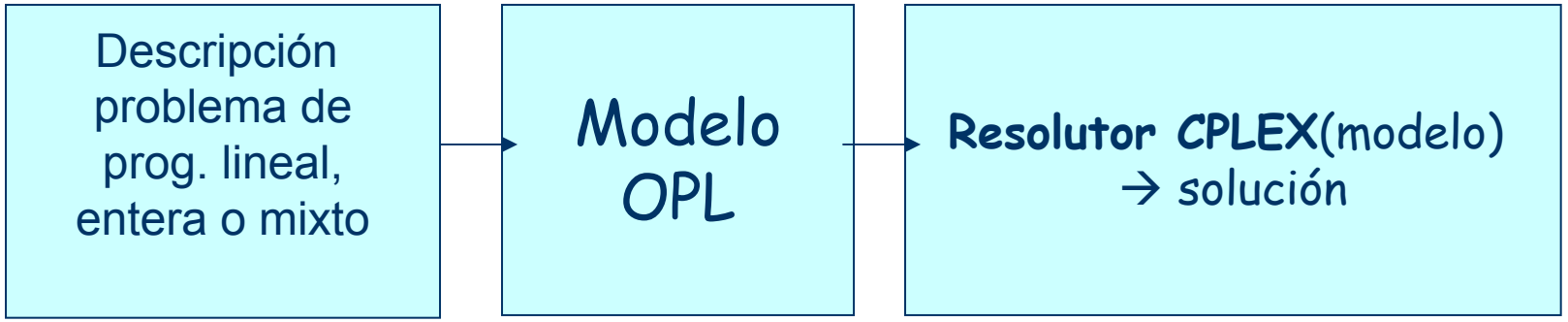
ILOG CPLEX

Problemas de:
programación lineal,
entera y lineal-entera

Ciclo de vida



Índice

- Ejemplo sencillo: LP, IP, MILP
  - Etapa resolución
- Ejemplos más complicados: LP, IP, MILP
  - Etapa modelado

Ejemplo sencillo

Descripción

Industria con stock de N, H y Cl

Se combinan para formar NH_3 y NH_4Cl.

NH_3 y NH_4Cl dan distintos beneficios.

Con el stock que tenemos, ¿qué producción nos dará más beneficio?

Modelo. Datos del problema

```
enum Componentes { N, H, Cl };
enum Productos { NH3, NH4Cl };

int stock[Componentes] = [7, 24, 4];

int requieren[Productos, Componentes] = [    [1, 3, 0],
                                             [1, 4, 1]    ];
float+ beneficio[Productos] = [30, 40];
```

Modelo. Variables de decisión

- Programación lineal

  var float x1;

  var float x2;

- Programación entera

  var int x1 in minint..maxint;

  var int x2 in minint..maxint;

- Programación entera-lineal mixta

  var int x1 in minint..maxint;

  var float x2;

Modelo. Función de óptimo

maximize

30*x1 + 40*x2

Modelo. Restricciones

```
subject to {
   1*x1 + 1*x2 <= 7;
   3*x1 + 4*x2 <= 24;
   0*x1 + 1*x2 <= 4;

   x1 >= 0;
   x2 >= 0;
};
```

Programación lineal

Resolución usando CPLEX

1. Región factible.
2. Evaluar vértices.

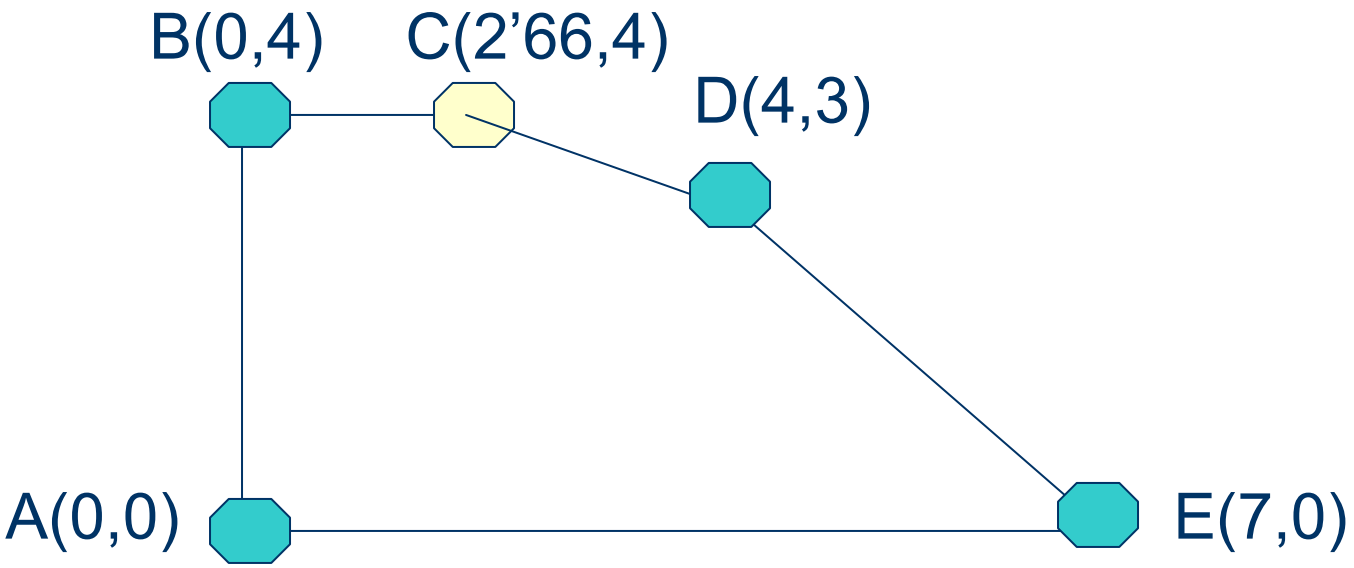
Ideas apoyo

- Restricciones redundantes
- Posibilidad de ∞ soluciones
- Elegir entre Simplex y Barrera.

  “ setting Lpmethod = barrier; “

Programación lineal

Resolvemos el área usando simplex:



Resultados OPL

Solutions

Optimal Solution with Objective Value: 240.0000

x1 = 2.6667

x2 = 4.0000

Log

Running

linear programming

displaying solution ...

Resultados OPL

CPLEX

Utilizó Simplex.

Variables: 2.

Constraints: 5.

Solver Memory: 171860

Solving time: 91.95

Programación entera

Discusión…

1. Todas las variables son enteras
2. OPL exige rango para var int…

Conclusión:

¿Programación entera = dominios finitos?

¡No!

=, <=, >= | <>, <, >

Programación entera

```
var int x1, x2 in minint..maxint;

maximize
  3*x1 + 2*x2
subject to {
  x1 + x2 < 4;
  2*x1 + x2 < 5;
  -x1 + 4*x2 > 2;
  x1 >= 0;
  x2 >= 0;
};
```

Resultados OPL

Solutions

Optimal Solution with Objective Value: 7

x1 = 1, x2 = 2

Log

Running

Solver

displaying solution ...

Solver

Variables: 2 Constraints: 5

Failures: 0 Choice Points: 3

Solver memory: 163840

Solving time: 0.24

Programación entera

```
var int x1, x2 in minint..maxint;

maximize
  3*x1 + 2*x2
subject to {
  x1 + x2 <= 4;
  2*x1 + x2 <= 5;
  -x1 + 4*x2 >= 2;
  x1 >= 0;
  x2 >= 0;
};
```

Resultados OPL

Solutions

Optimal Solution with Objective Value: 9

x1 = 1, x2 = 3

Log

Running

integer programming (CPLEX MIP)

displaying solution ...

CPLEX

Variables: 2

Constraints: 5

S.memory: 171860

Solving time: 0.92

Modelo. Programación entera

```
var int x1 in minint..maxint;
var int x2 in minint..maxint;

subject to {
   1*x1 + 1*x2 <= 7;
   3*x1 + 4*x2 <= 24;
   0*x1 + 1*x2 <= 4;
   x1 >= 0;
   x2 >= 0;
};
```

Programación entera

Resolución usando CPLEX

1. Región factible.
2. Evaluar vértices.
3. Solución no válida

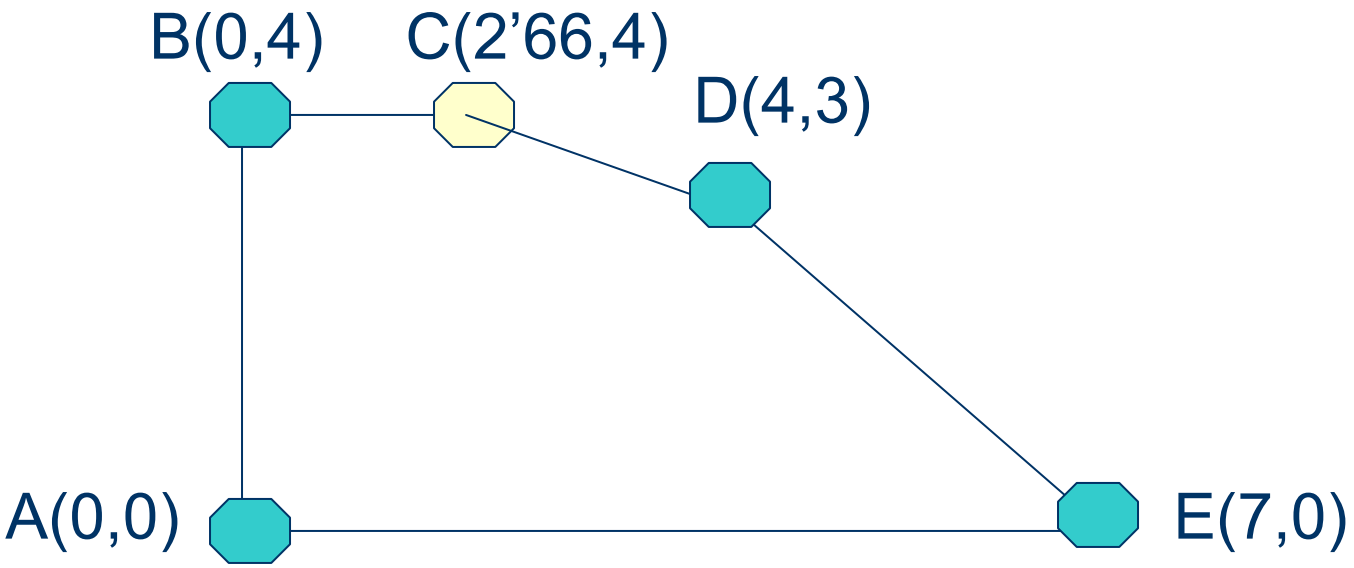
Idea…

¿Nueva técnica para Prog.Entera?

¡Reutilizar lo que ya funciona: Branch&Bound!

Árbol Branch&Bound

Resolvemos el área usando simplex:



Árbol Branch&Bound

Encontramos la solución:

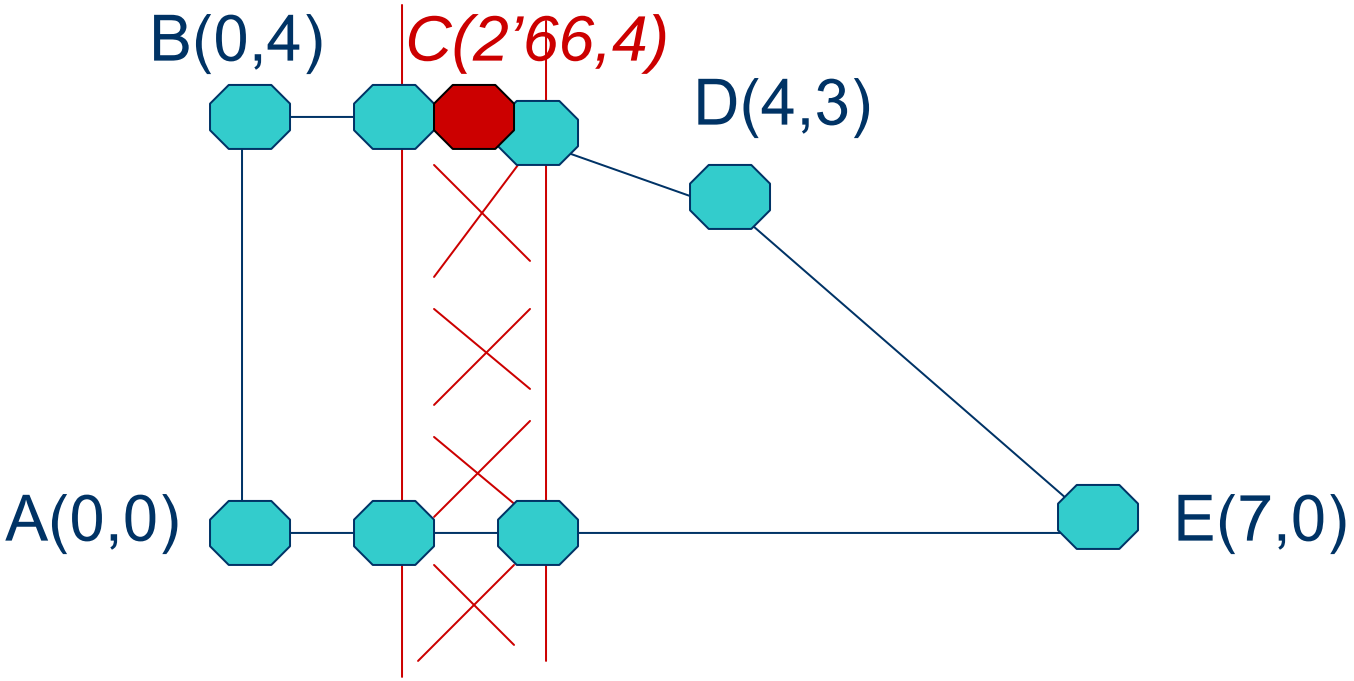
X1 = 2.66 X2 = 4.00
240.00

¡No es entera!

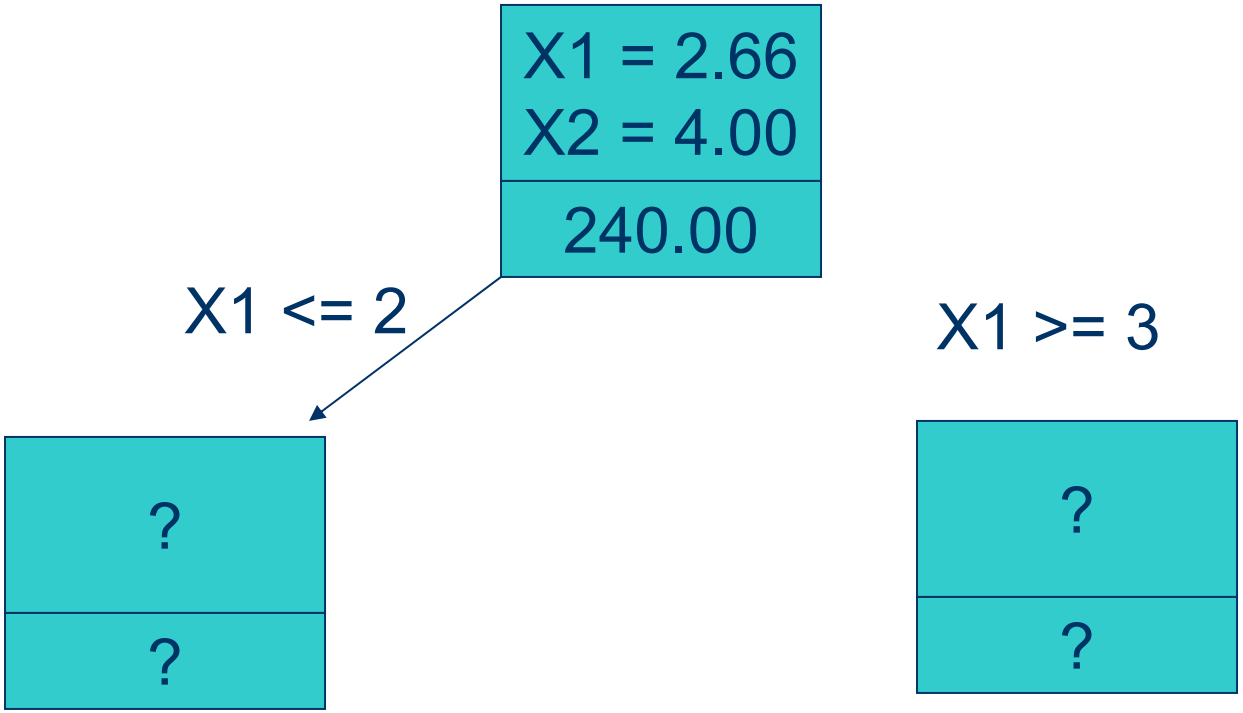
Bifurcar sobre X1

Árbol Branch&Bound

Con la bifurcación sobre X1:

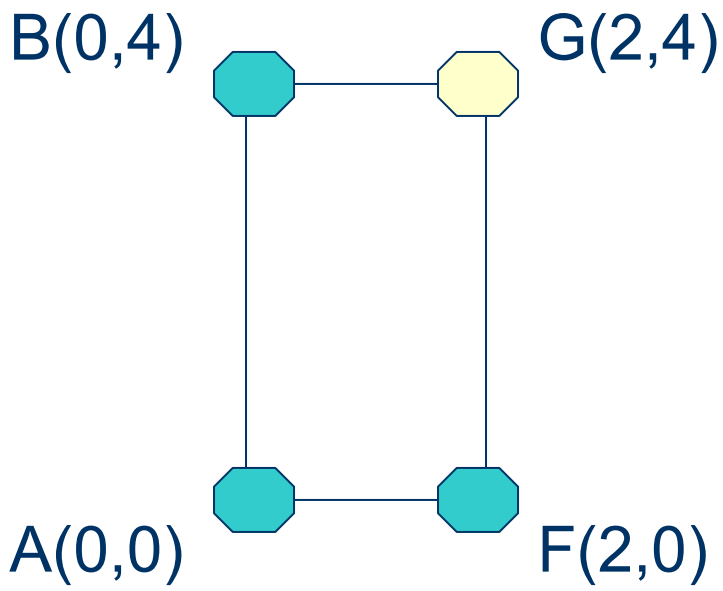


Árbol Branch&Bound

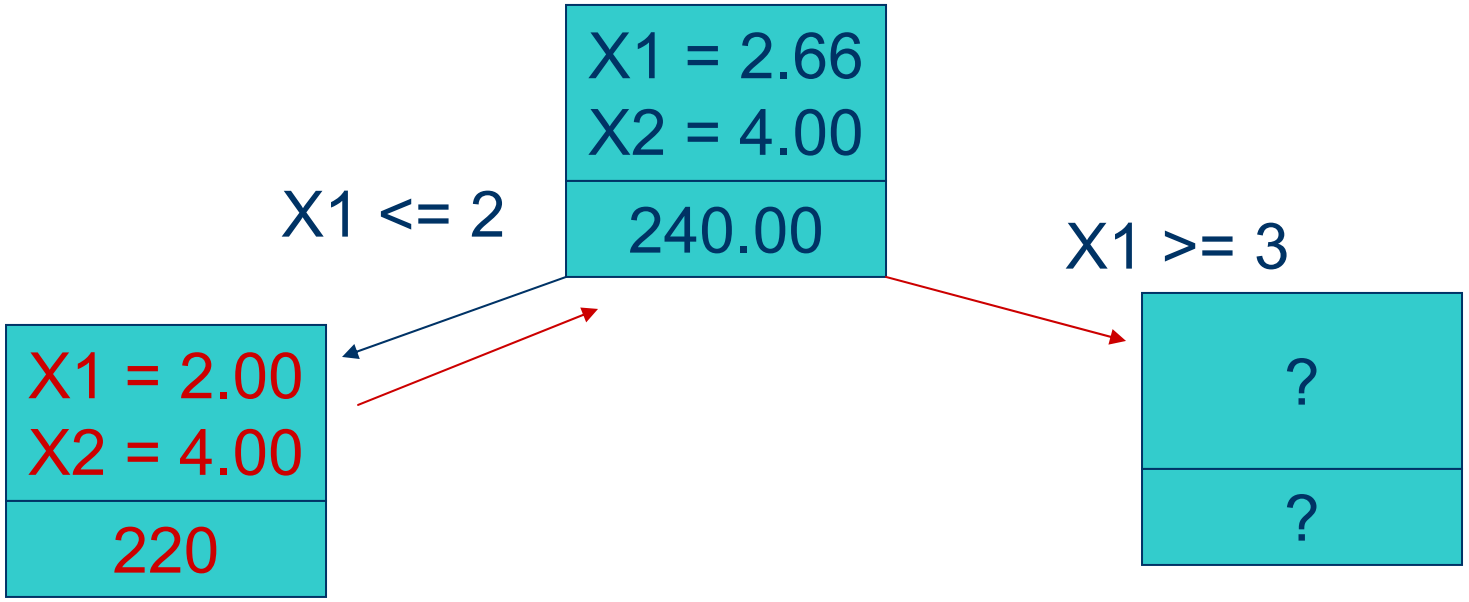


Árbol Branch&Bound

Resolvemos el área usando simplex:



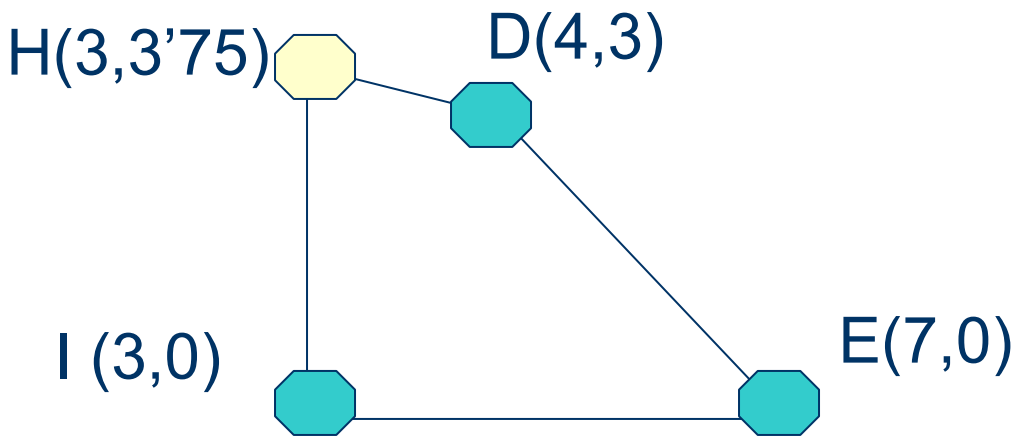
Árbol Branch&Bound



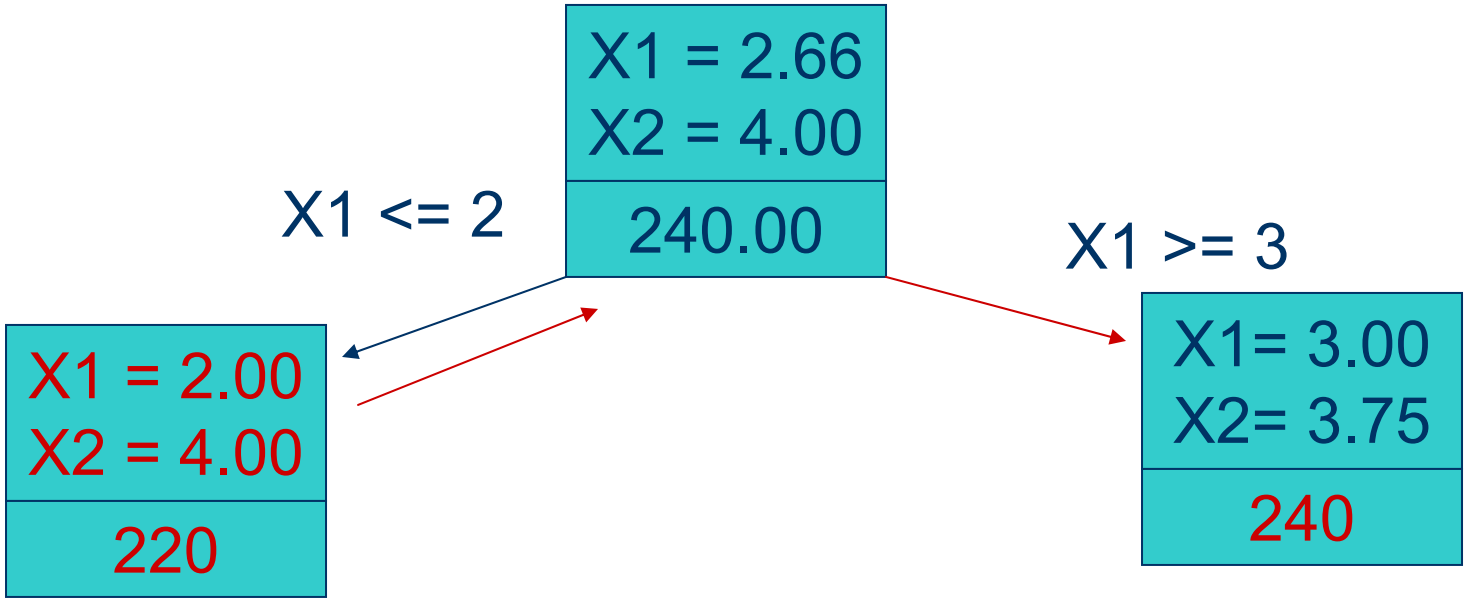
¡Solución entera!
Guardamos óptimo

Árbol Branch&Bound

Resolvemos el área usando simplex:



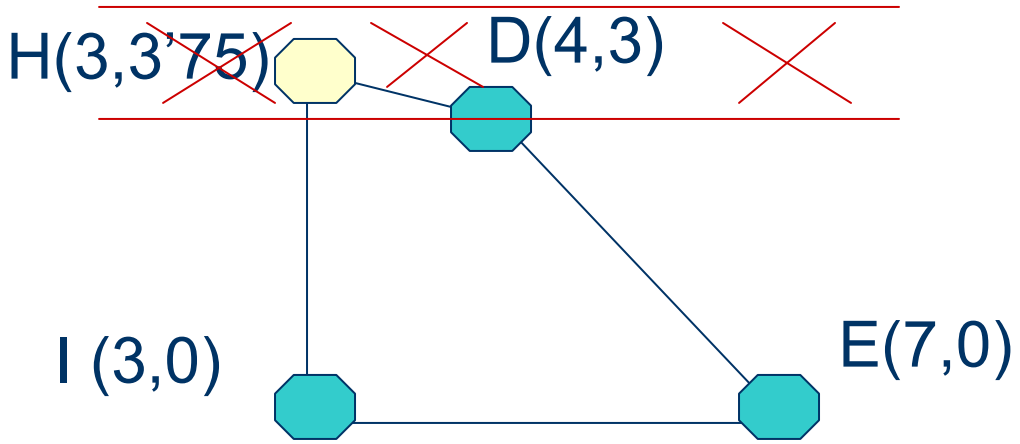
Árbol Branch&Bound



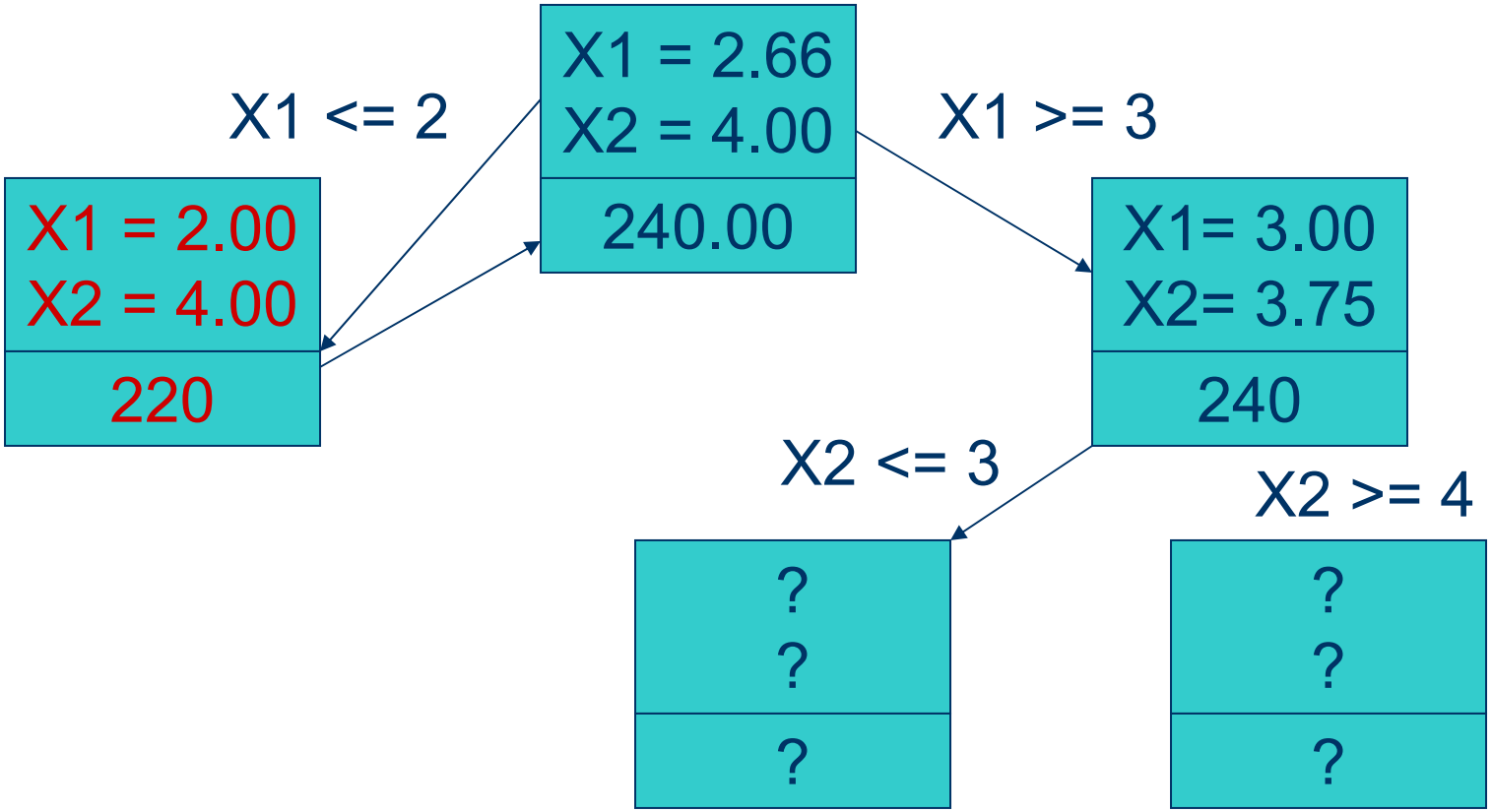
¡Supera, pero no es entera.
Bifurcar sobre X2

Árbol Branch&Bound

Con la bifurcación sobre X2:

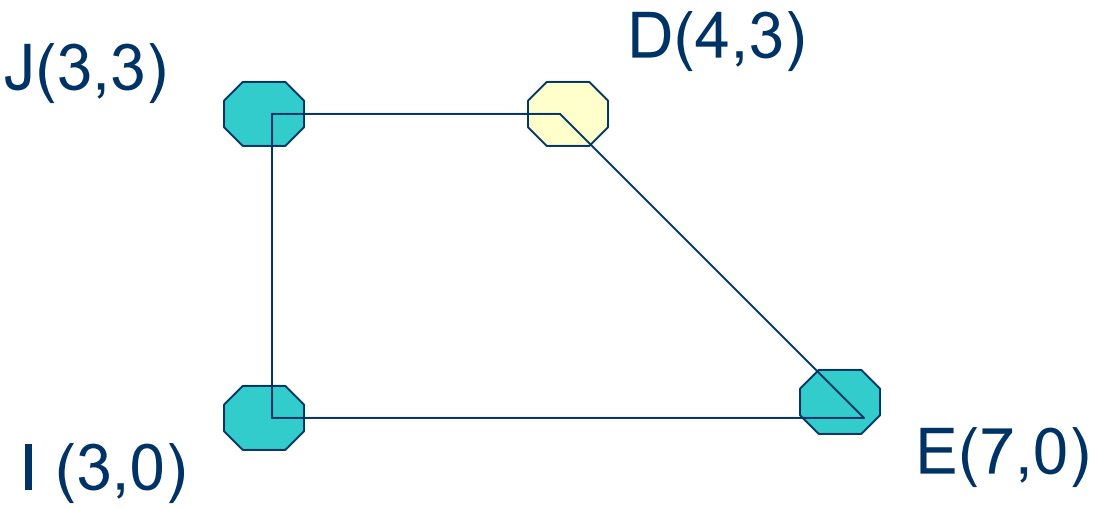


Árbol Branch&Bound

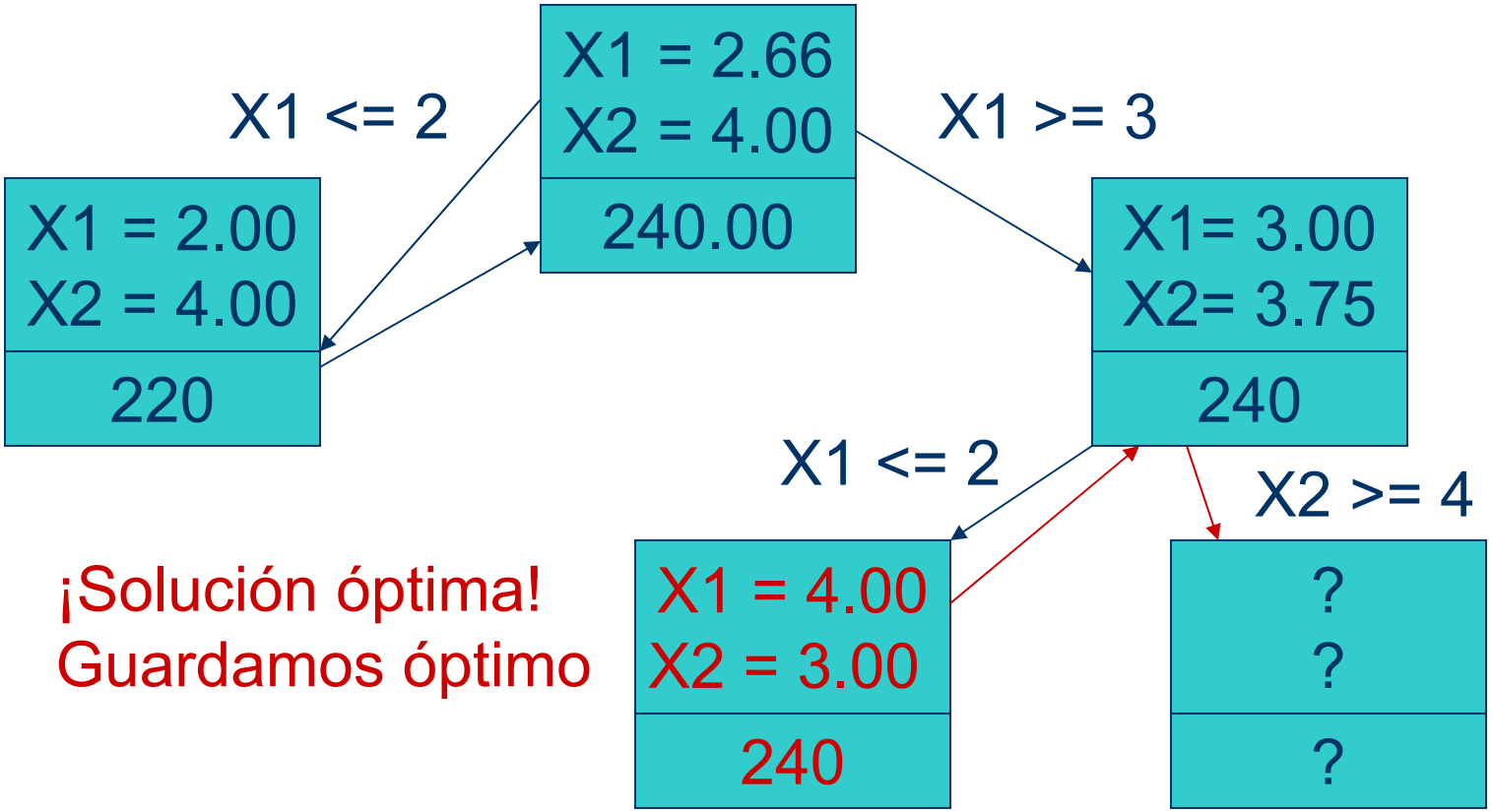


Árbol Branch&Bound

Resolvemos el área usando simplex:



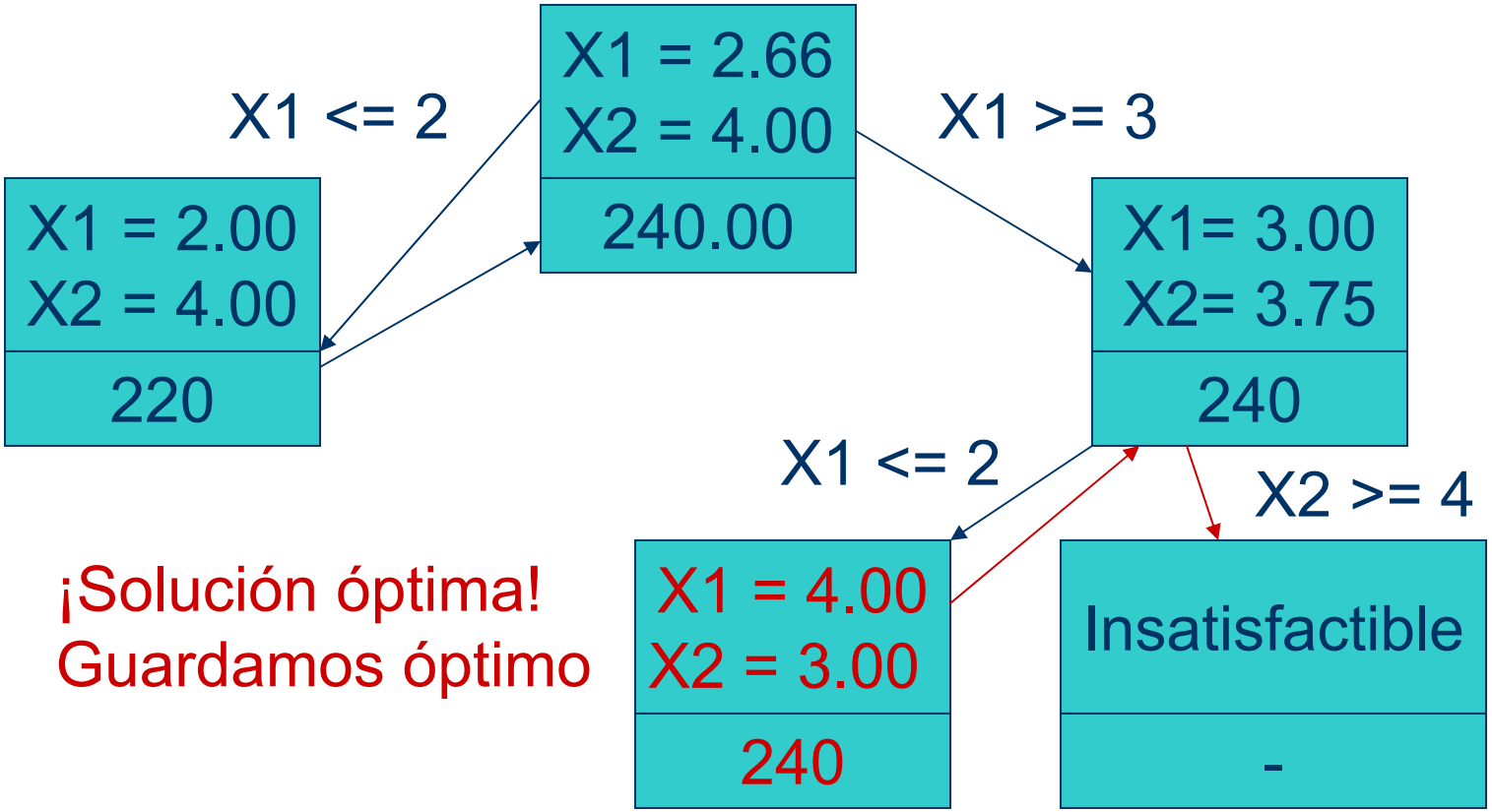
Árbol Branch&Bound



Árbol Branch&Bound

- *Para X1>= 3 y X2 >=4 no existe ninguna solución ya que nos hemos salido del área satisfactible*

Árbol Branch&Bound



Árbol Branch&Bound

¡La solución óptima es por lo tanto!

X1 = 4.00 X2 = 3.00
240

Resultados OPL

Solutions

Optimal Solution with Objective Value: 240

x1 = 4

x2 = 3

Log

Running

integer programming (CPLEX MIP)

displaying solution ...

Resultados OPL

CPLEX

Variables: 2.

Constraints: 5.

Solver Memory: 171860

Solving time: 0.03

Programación entera

Procedimientos de búsqueda.

¿Qué información añadir en el modelo OPL?

- Que variable poner en cada nivel del árbol
- Orden de exploración en los hijos

Programación entera

```
setting mipsearch{
    forall( v in vars)
        setPriority(v, PRIORIDAD);
        setBranchingDirection(v, branchDir*);
}
```

**Low, Up, Global*

Modelo. Programación entera mixta

```
var int x1 in minint..maxint;
var float x2;

subject to {
   1*x1 + 1*x2 <= 7;
   3*x1 + 4*x2 <= 24;
   0*x1 + 1*x2 <= 4;
   x1 >= 0;
   x2 >= 0;
};
```

Programación lineal-entera

Resolución usando CPLEX

1. Región factible.
2. Evaluar vértices.
3. Relajar problema + Branch&Bound

No bifurcar sobre variables float :-)

Resultados OPL

Solutions

Optimal Solution with
Objective Value: 240.0000
x1 = 4
x2 = 3.0000

CPLEX

Variables: 3
Restricciones: 6
Solver Memory: 171860
Solving time: 0.00

Log

Running
mixed integer programming
(CPLEX MIP)
displaying solution ...

Ejemplos modelado

- Programación lineal →

  Problema del transporte.

- Programación entera →

  Problema de la mochila.

  Problema del producto.

- Programación lineal-entera mixta →

  Problema de la receta.

LP → Problema del transporte

Teoría de fondo.

- Tamaño de los problemas combinatorios.

  Salto entre tamaño máximo LP – IP.
- Dentro de LP…

  ¿Cómo aumentar la eficiencia?

  Explotar la dispersión de datos.

Descripción

- Tenemos un conjunto de ciudades Gallegas y un conjunto de productos del mar.
- Cada ciudad produce y demanda una cierta cantidad de cada producto.
- Todas las ciudades están conectadas por carretera.

Descripción

- Tabla de coste depende de la ciudad origen, destino y producto transportado.
- Gobierno impone un límite en el transporte total (∑ productos) que se puede realizar entre cada par de ciudades.
- Minimizar el coste total del transporte.

1. Transporte gallego sin explotar

Consideramos que todas las ciudades están conectadas y que toda ciudad transporta todo producto a las demás ciudades.

Modelado. Datos del problema

```
enum Ciudades ...;
enum Productos ...;
float+ maximo = ...;

float+ suministra[Productos,Ciudades] = ...;
float+ demanda[Productos,Ciudades] = ...;
float+ cuesta[Productos,Ciudades,Ciudades] = ...;
```

Modelado. Datos del problema

Ciudades = { A_coruna, Lugo, Orense, Pontevedra,
Cebreiro, Melide, Santiago, Viveiro, Ortigueira,
Ribadiso };

Productos = { Centollos, Pulpo, Vieras };

maximo = 625;

Modelado. Datos del problema

```
suministra = #[
                Centollos: #[ A_coruna: 400, Lugo: 700,
                              Orense: 800, Pontevedra: 0,
                              Cebreiro: 0, Melide: 0,
                              Santiago: 0, Viveiro: 0,
                              Ortigueira: 0, Ribadiso: 0 ]#
                Pulpo: #[...]#
                Vieras: #[...]#
                 ]# ;
```

Modelado. Datos del problema

```
demanda = #[
                Centollos: #[ A_coruna: 0, Lugo: 0,
                              Orense: 0, Pontevedra: 300,
                              Cebreiro: 300, Melide: 100,
                              Santiago: 75, Viveiro: 650,
                            Ortigueira: 225, Ribadiso: 250 ]#
                Pulpo: #[...]#
                Vieras: #[...]#
                ]# ;
```

Modelado. Datos del problema

```
cuesta = #[
    Centollos: #[
            A_coruna:  #[ A_coruna: 0, Lugo: 0, Orense: 0, Melide: 8,
                          Pontevedra: 30, Cebreiro: 10, Ortigueira: 71,
                          Santiago: 10, Viveiro: 11, Ribadiso: 6 ]#
             Lugo: #[ ...]#
             ...
                    ]#
    Vieras: #[...]#
    Pulpos: #[...]#
  ]#
```

Modelado. Datos del problema

El problema exige a los datos de entrada cumplan que para cada producto la suma total demandada sea igual a la suma total producida.

```
assert
    forall(p in Productos)
        sum(o in Ciudades) suministra[p,o] =
        sum(d in Ciudades) demanda[p,d];
```

Variables de decisión

Solución al problema:

"Cuánta cantidad de producto transportar entre cada ciudad suministradora y consumidora."

var float+ trans[Productos,Ciudades,Ciudades];

Función de óptimo

Minimizar el coste de transportar todos los productos entre todas las ciudades.

```
minimize
    sum(p in Productos & o,d in Ciudades)
        cuesta[p,o,d] * trans[p,o,d]
```

Restricciones

- Una ciudad ‘o’ transporta una cantidad de producto ‘p’ a cada ciudad ‘c’, en total transportará ‘y’.
- Esa ciudad produce ‘x’ cantidad de ese producto.
- x = y

```
forall(p in Productos & o in Ciudades)
    sum(d in Ciudades) trans[p,o,d] = suministra[p,o];
```

Restricciones

- Una ciudad ‘d’ recibirá una cantidad de producto ‘p’ de cada ciudad ‘c’, en total recibirá ‘y’.
- Esa ciudad tiene una demanda ‘x’ de ese producto.
- x = y

```
forall(p in Productos & d in Ciudades)
    sum(o in Ciudades) trans[p,o,d] =  demanda[p,d];
```

Restricciones

- Una ciudad ‘o’ transportará una cantidad ‘k’ de cada producto a otra ciudad ‘d’, en total transportará ‘y’.
- Hay un límite máximo de transporte entre ciudades
- y <= limite

```
forall(o, d in Ciudades)
    sum(p in Productos) trans[p,o,d] <= maximo;
```

Mostrar resultados

display cuesta;

display trans;

Resultados OPL

Solutions

Optimal Solution with Objective Value: 199500.0000

trans[Centollos,Lugo,Orense] = 0.0000

trans[Centollos,Lugo,Pontevedra] = 225.0000

trans[Centollos,Lugo,Cebreiro] = 0.0000

trans[Centollos,Lugo,Melide] = 0.0000…

Log

Running

linear programming

displaying solution ...

CPLEX

Variables: 300 Restricciones: 160

Solver memory:344720 Solving Time: 0.02

Usó simplex

2. Transporte gallego optimizado

¿Cómo explotar la estructura del problema?

- Extraer de la matriz cuesta[p,o,d] que elementos son <> 0 → Rutas
- Extraer de cada ruta:
  - ✓ La pareja de ciudades → Conexiones
  - ✓ La pareja producto, origen → Suministra
  - ✓ La pareja producto, destino → Demanda

Modelado. Datos del problema

- struct Conexion {
  Ciudad o;
  Ciudad d;
  };
- struct Ruta {
  Producto p;
  Conexion e;
  };
- {Ruta} Rutas = ...;
- float+ cuesta[Rutas] = ...;

Modelo. Datos del problema

Rutas = {

<Centollos, <A_coruna, Pontevedra> >,

<Centollos, <A_coruna, Cebreiro> >,

<Centollos, <A_coruna, Melide> >,

<Centollos, <A_coruna, Santiago> >,

<Centollos, <A_coruna, Viveiro> >, … };

Modelo. Datos del problema

- Extraer de cada ruta cada pareja de ciudades conectadas.

  {Conexion} Conexiones = { c | <p,c> in Rutas };

- ¿Cómo hace el filtrado?
- ☒ Multiconjunto
- ☑ Conjunto

Modelo. Datos del problema

- Extraer las parejas <producto, ciudad origen>

```
struct Suministra {
        Producto p;
        Ciudad o;
};
{Suministra} Suministros = {<p,c.o>|<p,c> in Rutas };
float+ suministra[Suministros] = ...;
```

Modelo. Datos del problema

```
suministra = #[
        <Centollos, A_coruna>: 400
        <Pulpo, A_coruna>: 800
        <Vieras, A_coruna>: 200
        <Centollos, Lugo>: 700
        <Pulpo, Lugo>: 1600
        <Vieras, Lugo>: 300
        <Centollos, Orense>: 800
        <Pulpo, Orense>: 1800
        <Vieras, Orense>: 300
        ]#;
```

Modelo. Datos del problema

- Extraer las parejas <producto, ciudad destino>

```
struct Demanda {
        Producto p;
        Ciudad o;
};
{Demanda} Demandas = {<p,c.d>|<p,c> in Rutas };
float+ demanda[Demandas] = ...;
```

Modelo. Datos del problema

```
demanda = #[
        <Centollos, Pontevedra>: 300
        <Pulpo, Pontevedra>: 500
        <Vieras, Pontevedra>: 100
        <Centollos, Cebreiro>: 300
        <Pulpo, Cebreiro>: 750
        <Vieras, Cebreiro>: 100
        <Centollos, Melide>: 100
       ...
        ]# ;
```

Modelo. Datos del problema

¿Cómo explotar la estructura del problema?

- Para cada producto saber…
  - ¿Qué ciudades lo producen?
  - ¿Qué ciudades lo demandan?
  - ¿Qué conexiones concretas se establecen?

{Ciudad} orig[p in Producto] = { c.o | <p,c> in Rutas };
{Ciudad} dest[p in Producto] = { c.d | <p,c> in Rutas };
{Conexion} CP[p in Producto] = { c | <p,c> in Rutas };

Modelo. Datos del problema

El problema exige a los datos de entrada cumplan que para cada producto la suma total demandada sea igual a la suma total producida.

assert forall(p in Producto)

sum(o in orig[p]) suministra[<p,o>] = sum(d in dest[p]) demanda[<p,d>];

Variables de decisión

var float+ trans[Rutas];

1. Transporte sin optimizar → 300 variables.
2. Transporte optimizado → 63 variables.

Función de óptimo

```
minimize
    sum(l in Rutas) cuesta[l] * trans[l]
```

Restricciones

- Una ciudad ‘o’ transporta una cantidad de producto ‘p’ a cada ciudad ‘c’, en total transportará ‘y’.
- Esa ciudad produce ‘x’ cantidad de ese producto.
- x = y

```
forall(p in Producto & o in orig[p])
    sum(<o,d> in CP[p]) trans[< p,<o,d> >] =
          suministra[<p,o>];
```

Restricciones

- Una ciudad ‘d’ recibirá una cantidad de producto ‘p’ de cada ciudad ‘c’, en total recibirá ‘y’.
- Esa ciudad tiene una demanda ‘x’ de ese producto.
- x = y

```
forall(p in Producto & d in dest[p])
    sum(<o,d> in CP[p]) trans[< p,<o,d> >] =
          demanda[<p,d>];
```

Restricciones

- Una ciudad ‘o’ transportará una cantidad ‘k’ de cada producto a otra ciudad ‘d’, en total transportará ‘y’.
- Hay un límite máximo de transporte entre ciudades
- y <= limite

forall(c in Conexiones)

sum(<p,c> in Rutas) trans[<p,c>] <= maximo;

Resultados OPL

Solutions

Optimal Solution with Objective Value: 199500.0000

trans[#<p:Centollos,e:#<o:A_coruna,d:Viveiro>#>#] = 400.0000

trans[#<p:Centollos,e:#<o:A_coruna,d:Ortigueira>#>#] = 0.0000

trans[#<p:Centollos,e:#<o:Lugo,d:Pontevedra>#>#] = 225.0000

…

Log

Running

linear programming

displaying solution ...

CPLEX

Variables: 63 Restricciones: 51

Solver memory: 224120Solving time: 0.01

Usó simplex

Resultados OPL

Solutions

Optimal Solution with Objective Value: 199500.0000

trans[Centollos,Lugo,Orense] = 0.0000

trans[Centollos,Lugo,Pontevedra] = 225.0000

trans[Centollos,Lugo,Cebreiro] = 0.0000

trans[Centollos,Lugo,Melide] = 0.0000…

Log

Running

linear programming

displaying solution ...

CPLEX

Variables: 300 Restricciones: 160

Solver memory:344720 Solving Time: 0.02

Usó simplex

Ejemplos modelado

- Programación lineal →

  Problema del transporte.

- Programación entera →

  Problema de la mochila.

  Problema del producto.

- Programación lineal-entera mixta →

  Problema de la receta.

IP → Problema de la mochila

Descripción

- Tenemos una mochila con una cierta capacidad: (Peso máximo, Volumen, …) y varias clases de objetos: (Libros, botas, bocatas, …).
- Cada objeto tiene un valor y un cierto ‘uso’ de la mochila(peso, volumen,…)

- Maximizar el valor de los objetos elegidos respetando la capacidad que tiene la mochila.

Modelado. Datos del problema

```
int num_objetos = ...;
int num_caracteristicas = ...;
range objetos 1..num_objetos;
range caracteristicas 1..num_caracteristicas;
//Instancia
num_objetos = 12;
num_caracteristicas = 7;
```

Modelo. Datos del problema

```
int maximo[caracteristicas] = ...;
int valor[objetos] = ...;

int max_objetos = max(c in caracteristicas) maximo[c];

//Instancia
maximo = [ 18209, 7692, 1333, 924, 26638, 61188, 13360 ];
valor = [ 96, 76, 56, 11, 86, 10, 66, 86, 83, 12, 9, 81 ];
```

Modelo. Datos del problema

```
int uso[caracteristicas, objetos] = ...;
//Instancia
uso = [   [ 19,   1,  10,  1,   1,  14, 152, 11,  1,   1, 1, 1 ],
          [  0,   4,  53,  0,   0,  80,   0,  4,  5,   0, 0, 0 ],
          [  4, 660,   3,  0,  30,   0,   3,  0,  4,  90, 0, 0],
          [  7,   0,  18,  6, 770, 330,   7,  0,  0,   6, 0, 0],
          [  0,  20,   0,  4,  52,   3,   0,  0,  0,   5, 4, 0],
          [  0,   0,  40, 70,   4,  63,   0,  0, 60,   0, 4, 0],
          [  0,  32,   0,  0,   0,   5,   0,  3,  0, 660, 0, 9]   ];
```

Variables de decisión

Cuantos objetos guardar de cada tipo

var int guarda[objetos] in 0..max_objetos;

Función de óptimo

Maximizar el beneficio obtenido por los objetos llevados.

```
maximize
    sum(o in objetos) valor[o] * guarda[o]
```

Restricciones

El conjunto de objetos elegidos no debe sobrepasar (peso, volumen, ...) de la mochila.

```
subject to
    forall(c in caracteristicas)
        sum(o in objetos) uso[c, o] * guarda[o] <=
        maximo[c];
```

Resultados OPL

Solutions

Optimal Solution with Objective Value: 261922

guarda[1] = 0,	guarda[2] = 0,
guarda[3] = 0,	guarda[4] = 154,
guarda[5] = 0,	guarda[6] = 0,
guarda[7] = 0,	guarda[8] = 913 ,
guarda[9] = 333,	guarda[10] = 0,
guarda[11] = 6499,	guarda[12] = 1180

Log

Running integer programming (CPLEX MIP)

displaying solution ...

CPLEX Usó MIP

Variables: 12 Restricciones: 7

Solver memory: 191960

Solving time: 0.05

Ejemplos modelado

- Programación lineal →

  Problema del transporte.

- Programación entera →

  Problema de la mochila.

  Problema del producto.

- Programación lineal-entera mixta →

  Problema de la receta.

Problema del producto

Descripción

Para elaborar un producto hacen falta un número *n* de tareas.

Contratamos para ello a un número *m* de trabajadores.

Cada trabajador está capacitado para realizar un subconjunto de las *n* tareas.

Cada trabajador tiene un sueldo.

Problema del producto

Descripción

Contratar a los trabajadores necesarios para poder realizar todas las tareas. Minimizar el coste de contratación

Modelo. Datos del problema

Identificamos a cada trabajador con un entero.

Creamos un rango para identificar a cada uno de ellos.

```
int num_Trabajadores = ...;
range Trabajadores 1..num_Trabajadores;
```

Modelo. Datos del problema

Tendremos también un conjunto de tareas.

```
enum Tareas ...;
//Instancia
Tareas = { albanileria, carpinteria, tuberias, techos,
    electricidad, calefaccion, insonorizacion, tejados,
    pintura, ventanas, fachada, jardines,
    garaje, parking, mudanza };
```

Modelo. Datos del problema

El sueldo y capacidad de cada trabajador…

{Trabajadores} pueden_hacer[Tareas] = ...;

int sueldo[Trabajadores] = ...;

Modelo. Datos del problema

```
//Instacia
pueden_hacer = [
   { 1  9 19  22  25  28  31 }
   { 2 12 15 19 21 23 27 29 30 31 32 }
   { 3 10 19 24 26 30 32 }
   { 4 21 25 28 32 }
   { 5 11 16 22 23 27 31 }
   { 6 20 24 26 30 32 }
    …
];
sueldo = [ 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 6 6 6 7
     8 9 ];
```

Modelo. Variables de decisión

La solución será saber para cada trabajador si lo cogemos o no.

var int contratamos[Trabajadores] in 0..1;

Modelo. Función de óptimo

Minimizar el coste de contratación

```
minimize
   sum(c in Trabajadores)sueldo[c]*contratamos[c]
```

Modelo. Restricciones

Que toda tarea pueda ser realizada al menos por uno de los trabajadores contratados.

```
forall(j in Tareas)
    sum(c in pueden_hacer[j]) contratamos[c] >= 1;
```

Resultados OPL

Solutions

Optimal Solution with Objective Value: 14

contratamos[23] = 1

contratamos[25] = 1

contratamos[26] = 1

Log

Running

integer programming (CPLEX MIP)

displaying solution ...

CPLEX

Variables: 32

Restricciones: 15

Solver memory: 204020

Solving time: 1.81

Ejemplos modelado

- Programación lineal →

  Problema del transporte.

- Programación entera →

  Problema de la mochila.

  Problema del producto.

- Programación lineal-entera mixta →

  Problema de la receta.

MILP → Problema de la receta

Descripción

- Preparación de la poción mágica de Panoramix.
- La cacerola tiene un volumen v. Hay que llenarla.

¿Con qué?

- Líquidos: Agua, Zumo, Gatorade.
- Árboles: Raíces y Hojas.
- Brebajes: BrebajeA y BrebajeB.
- Pájaros: Gorrión.

MILP → Problema de la receta

- La mezcla debe ser buena.

  Porcentaje mínimo y máximo de cada ingrediente.

- Encontrar cada ingrediente tiene un coste:
  - Líquidos: tiempo por litro.
  - Árbol: tiempo extracción por kg (1Kg = 1L)
  - Brebajes: tiempo por litro
  - Pájaros: tiempo por unidad.

MILP → Problema de la receta

Objetivo: minimizar el tiempo en conseguir todos los ingredientes para una poción cuya mezcla sea buena.

Modelo. Datos del problema

```
enum Liquidos ...;
enum Arboles ...;
enum Brevajes ...;
enum Pajaros ...;
//Instancia
Liquidos = {Agua, Zumo, Gatorade};
Arboles = {Raices, Hojas};
Brebajes = {Brebaje_A, Brebaje_B};
Pajaros = {Gorriones};
```

Modelo. Datos del problema

Mezcla buena

```
float+ minimo[Liquidos] = ...;
float+ maximo[Liquidos] = ...;
int+ volumen  = ...;
//Instancia
minimo = [0.05, 0.30, 0.60];
maximo = [0.10, 0.40, 0.80];
volumen  = 20;
```

Modelo. Datos del problema

Porcentaje de líquidos en Árbol, Brebajes y Pájaros.

```
float+ porc_arboles[Liquidos,Arboles] = ...;
float+ porc_brevajes[Liquidos,Brevajes] = ...;
float+ porc_pajaros[Liquidos,Pajaros] = ...;
//Instancia
porc_arboles = [[ 0.20, 0.01 ], [ 0.05, 0 ], [ 0.05, 0.30 ]];
porc_brebajes = [ [ 0 , 0.01 ], [ 0.60, 0 ], [ 0.40, 0.70 ] ];
porc_pajaros = [ [ 0.10 ], [ 0.45 ], [ 0.45 ] ];
```

Modelo. Datos del problema

```
Tiempo en encontrar cada cosa
float+ tiempo_liquidos[Liquidos] = ...;
float+ tiempo_arboles[Arboles] = ...;
float+ tiempo_brevajes[Brevajes] = ...;
float+ tiempo_pajaros[Pajaros] = ...;
//Instancia
tiempo_liquidos = [22, 10, 13];
tiempo_arboles = [6, 5];
tiempo_brevajes = [ 7, 8];
tiempo_pajaros = [ 9 ];
```

Variables de decisión

```
var float+ li[Liquidos];
var float+ ar[Arboles];
var float+ br[Brebajes];
var int+ pa[Pajaros] in 0..maxint;

var float+ mezcla[l in Liquidos];
```

Modelo. Función de óptimo

```
minimize
  sum(l in Liquidos) tiempo_liquidos[l] * li[l]
        +
  sum(a in Arboles)   tiempo_arboles[a]   * ar[a]
        +
  sum(b in Brebajes) tiempo_brebajes[b] * br[b]
        +
  sum(p in Pajaros)  tiempo_pajaros[p]  * pa[p]
```

Modelo. Restricciones

Qué la cantidad de cada litro sea calculada correctamente

```
forall(l in Liquidos)
    mezcla[l] =
        li[l] +
        sum(a in Arboles) porc_arboles[l,a] * ar[a] +
        sum(b in Brebajes) porc_brebajes[l,b] * br[b] +
        sum(p in Pajaros)  porc_pajaros[l,p] * pa[p];
```

Modelo. Restricciones

Que la mezcla respete el mínimo y máximo de cada líquido.

```
forall(l in Liquidos){
      mezcla[l] >= minimo[l] * volumen;
      mezcla[l] <= maximo[l] * volumen;
};
```

Modelo. Restricciones

Que el caldero esté lleno.

sum(l in Liquidos) mezcla[l] = volumen;

Resultados OPL

Solutions

Optimal Solution with Objective Value: 184.1171

li[Agua] = 0.0145, ar[Raices] = 0.0000,
li[Zumo] = 0.0000, ar[Hojas] = 0.0000,
li[Gatorade] = 0.0000, br[Brebaje_A] = 4.9167,
br[Brebaje_B] = 8.5476, pa[Gorriones] = 9

mezcla[Agua] = 1.0000, mezcla[Zumo] = 7.0000,
mezcla[Gatorade] = 12.0000

Resultados OPL

Log

Running

mixed integer programming (CPLEX MIP)

displaying solution ...

CPLEX

Usó MIP.

Variables: 12 Restricciones: 11

Solver memory: 183920 Solving time: 0.00

Fin