MA4702. Programación Lineal Mixta. 2020.
Profesor: José Soto
Fecha: 27 de Marzo 2020.

Cátedra 3

1. Descripción de Branch and Bound

Uno de los métodos más usados para resolver PLM y otros problemas de optimización es **branch and bound (BnB)**. Se basa en la siguiente idea:

> Sea $\{S_1, \ldots, S_k\}$ una partición del conjunto factible S de un problema de optimización $z^* = \max\{c^T x \colon x \in S\}$. Si $z_i^* = \max\{c^T x \colon x \in S_i\}$ entonces $z^* = \max_{i \in [k]} z_i^*$.

BnB consiste en particionar el conjunto factible S del problema original en conjuntos más pequeños y resolver luego $\max c^T x$ en cada subconjunto de manera recursiva. Si la recursión se pudiera llevar completamente, al final enumeraríamos todos los puntos factibles del problema. Esta idea tiene dos problemas. Primero, si el dominio es infinito entonces esto no es posible. Segundo, incluso si el dominio es finito, esto podría ser extremadamente lento y no difiere en nada de simplemente de simple fuerza bruta.

La idea es que BnB no explore todo el árbol de recursión sino que guarde cotas para los subproblemas que ya ha resuelto y, usando estas cotas determinar que no necesitamos resolver ciertos subproblemas.

En lo que sigue nos enfocaremos en PLM de la forma

$$(M) \quad \max c^T x$$
$$S\colon \begin{cases} Ax & \leq b \\ x & \in \mathbb{Z}^E \times \mathbb{R}^C \end{cases}$$

donde A, b, c son racionales. Llamemos $P = \{x \in \mathbb{R}^{E \times C}, Ax \leq b\}$ a la relajación lineal natural de S, luego $S \subseteq P$. Como los datos son racionales tenemos que el programa lineal relajado

$$(L) \quad \max c^T x$$
$$P\colon \begin{cases} Ax & \leq b \\ x & \in \mathbb{R}^{E \cup C} \end{cases}$$

es o bien **infactible**, o bien **factible no acotado** o bien **factible acotado** con solución óptima $\bar{x} \in P$ racional. En este caso llamamos $\bar{z} = c^T \bar{x}$ a su valor.

Llamemos (M_0) al problema inicial dado y **pediremos que** (P_0) **sea factible acotado**. Hacemos esto pues BnB *a secas* no es capaz de lidiar con problemas con relajación no acotada en la dirección de optimización.

Para resolver (M_0) se construye **iterativamente** un árbol $\mathcal{T}$ cuyos nodos son de la forma (M, B, s) con (M) un subproblema de (M_0), B una cota superior (optimista) del valor de (M_0), y s un **status** que puede ser **activo**, **ramificado**, **infactible**, **dominado** o **entero**. Tanto la cota como el status de un nodo puede cambiar a lo largo del algoritmo.

En el árbol siempre se tendrá que **la unión de los dominios de las hojas** es el **dominio** de la raíz. Además se mantiene globalmente una solución factible (inicialmente nula) x^* llamada **incumbente** que resulta ser la mejor solución encontrada hasta ahora y $z^* = c^T x^*$ su valor (también llamada **mejor cota inferior**). Inicialmente, el nodo raíz es $(M_0, B_0, \text{activo})$ donde M_0 es el PLM original

y B_0 es el valor $\bar{z}$ de su relajación. Además, x^* es nulo y $z^* = -\infty$. Durante el algoritmo BnB hay 2 procesos importantes: la **creación de un nodo** y la **ramificación de un nodo activo**.

Algorithm 1 Creación de un nodo (M)

1: Resolver la relajación lineal (P) de (M).
2: **if** (P) es infactible **then return** $(M, -\infty, \text{infactible})$.
3: **if** el óptimo $\bar{x}$ de (P) es factible para (M) **then**
4: **if** $\bar{z} \leq z^*$ **then return** $(M, \bar{z}, \text{entero})$
5: **if** $\bar{z} > z^*$ **then** $x^* \leftarrow \bar{x}$, $z^* \leftarrow \bar{z}$, **return** $(M, \bar{z}, \text{entero})$ ▷ actualizar incumbente
6: **if** el óptimo $\bar{x}$ de (P) es infactible para (M) **then**
7: **if** $\bar{z} \leq z^*$ **then return** $(M, \bar{z}, \text{dominado})$
8: **if** $\bar{z} > z^*$ **then return** $(M, \bar{z}, \text{activo})$

Notamos que si un nodo se declara entero, entonces conocemos su mejor valor factible. Si un nodo se declara dominado en su dominio S no pueden haber soluciones enteras mejores que el incumbente, por lo que no es necesario seguir procesándolo, al igual que si el nodo se declara infactible. Los únicos problemas que podrían tener soluciones enteras mejores que el incumbente actual son aquellos que están activos.

Por otro lado, ramificar un nodo activo $(M, B(M), \text{activo})$ consiste en:

Algorithm 2 Ramificar nodo $(M, B(M), \text{activo})$

1: Determinar $k \geq 2$ subproblemas (M_i) tales que la unión de sus dominios es el dominio de M.
2: Crear un nodo (M_i) para cada subproblema y colgarlo como hijo de (M).
3: Declarar el status de (M) como ramificado.

Hay varias formas de ramificar un nodo. Una forma estándar y simple es hacer **branching en una variable dada**.

Branching en una variable x_k Como el óptimo $\bar{x} \in P$ de (P) no es factible en (M) debe haber una coordenada $k \in E$ tal que la variable $\bar{x}_k$ es fraccional (pero que debería ser entera en (M)). Podemos **elegir** una coordenada y definir entonces dos PLM nuevos (M_1) y (M_2) como sigue:

$$S_1 = S \cap \{x\colon x_k \leq \lfloor \bar{x}_k \rfloor\}. \qquad S_2 = S \cap \{x\colon x_k \geq \lceil \bar{x}_k \rceil\}.$$
$$P_1 = P \cap \{x\colon x_k \leq \lfloor \bar{x}_k \rfloor\}. \qquad P_2 = P \cap \{x\colon x_k \geq \lceil \bar{x}_k \rceil\}.$$
$$(M_1)\colon \text{máx}\{c^T x\colon x \in S_1\}. \qquad (M_2)\colon \text{máx}\{c^T x\colon x \in S_2\}.$$
$$(L_1)\colon \text{máx}\{c^T x\colon x \in P_1\}. \qquad (L_2)\colon \text{máx}\{c^T x\colon x \in P_2\}.$$

Esto satisface las condiciones de la ramificación anterior. Ahora estamos listos para escribir el algoritmo de BnB completo. Como BnB es un método genérico hay algunas instrucciones (en rojo) que no están completamente descritas.

Algorithm 3 BnB

Ensure: Un PLM (M_0) **racional** con relajación (L_0) factible acotada.

$x^* \leftarrow$ NULL, Crear nodo (M_0) como raíz del árbol $\mathcal{T}$.
while existan nodos activos en $\mathcal{T}$ **do**
    Si se ha alcanzado un criterio de terminación temprana parar
    Elegir un nodo activo (M, B, activo) y ramificarlo.
    Actualizar las cotas $B(M')$ para cada nodo (M') en el camino entre (M) y la raíz (M_0), i.e.
        $B(M') \leftarrow \min\{B(M'), \max\{B(\tilde{M}) \colon (\tilde{M}) \text{ es hijo de } (M')\}\}$
    Si el incumbente cambió, **actualizar** todos los nodos activos que ahora estén dominados, i.e.
        Declarar todo (M', B', activo) con $B' \leq z^*$ como dominado.
Return x^*.

Discutiremos luego criterios de terminación temprana. Pero si en algún minuto necesitamos terminar, entonces observamos que el valor óptimo de M_0 siempre está en $[z^*, B(M_0)]$. La razón $\frac{B(M_0)-z^*}{z^*}$ se suele llamar el **GAP** de resolución (en dicho momento). Hagamos un ejemplo concreto de BnB usando solo branchings por variables.

$$M_0: \quad \max 3x + 5y$$

$$S_0: \begin{cases} 20y + 9x & \leq 74 \\ 25y + 18x & \leq 105 \\ x, y & \geq 0 \\ x, y & \in \mathbb{Z} \end{cases}$$

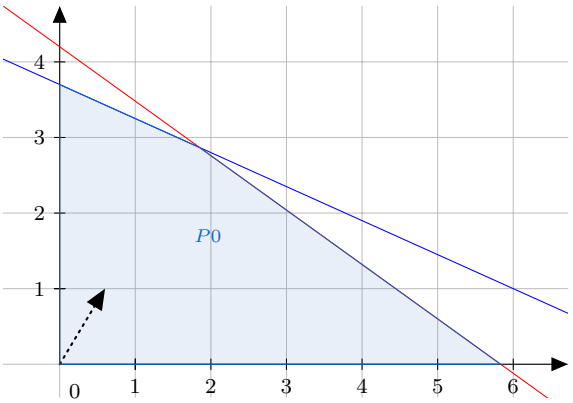
```
modelo= Model()
set_optimizer(modelo, Gurobi.Optimizer)
set_optimizer_attributes(modelo, "Presolve" =>
0,
"OutputFlag" => 0)
@variable(modelo, x>=0, base_name="x")
@variable(modelo, y>=0, base_name="y")
@constraint(modelo, rest1, 20y + 9x<=74)
@constraint(modelo, rest2, 25y+ 18x<=105)
@objective(modelo, Max, 3x+5y)
optimize!(modelo)
termination_status(modelo)

OPTIMAL::TerminationStatusCode = 1

println("z=",objective_value(modelo)," x=",
  value(modelo[:x]),"  y=", value(modelo[:y]))

z=19.799999999999997 x=2.0 y=2.76
```

Al resolver la relajación lineal anterior (por ejemplo con el solver GUROBI mediante JULIA, a la derecha), encontramos que el óptimo del PL asociado es $(x_0, y_0) \approx (1.8518, 2.866)$, $z \approx 19.888$ que no es entero. Luego el nodo raíz M_0 queda activo, con cota superior $B = 19.888$.

Estudiemos el árbol que BnB produce, por simplicidad anotemos en cada nodo además el punto fraccional óptimo de la relajación. Escribamos además sobre la raíz los datos actuales del incumbente.

$(x^*, y^*) = \text{NULL}, z^* = -\infty$

$(M_0, B = 19.888, \text{ACT})$
$(1.8518, 2.866), z_0 = 19.888$



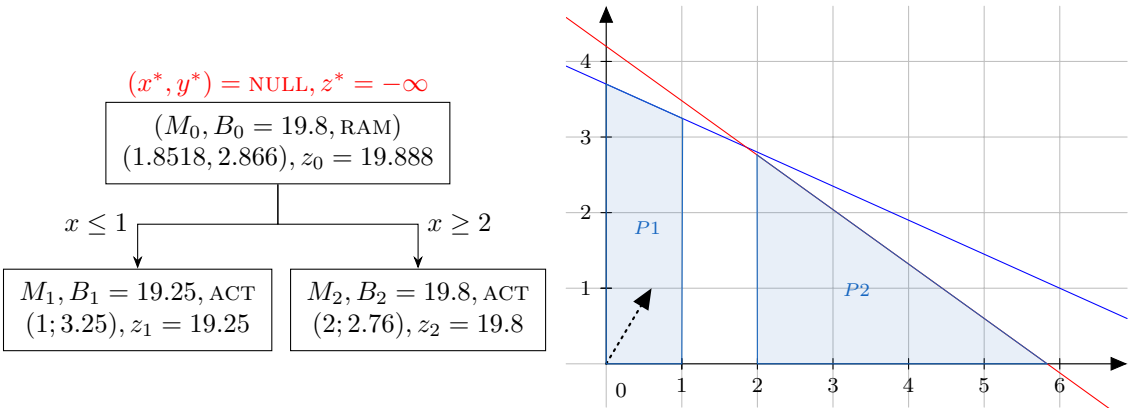
Como ambas coordenadas de (x_0, y_0) son fraccionales **podemos elegir una**, digamos x y **ramificar** de acuerdo a dicha variable, creando dos problemas M_1 y M_2, con conjuntos factibles $S_1 = S_0 \cap \{x \leq \lfloor x_0 \rfloor\}$ y $S_1 = S_0 \cap \{x \geq \lceil x_0 \rceil\}$.

$$M_1: \quad \text{máx } 3x + 5y \qquad M_2: \quad \text{máx } 3x + 5y$$

$$S_1: \begin{cases} 20y + 9x & \leq 74 \\ 25y + 18x & \leq 105 \\ x & \leq 1 \\ x, y & \geq 0 \\ x, y & \in \mathbb{Z} \end{cases} \qquad S_2: \begin{cases} 20y + 9x & \leq 74 \\ 25y + 18x & \leq 105 \\ x & \geq 2 \\ x, y & \geq 0 \\ x, y & \in \mathbb{Z} \end{cases}$$

```
#modelo M1
set_upper_bound(modelo[:x], 1)
optimize!(modelo)
termination_status(modelo)

OPTIMAL::TerminationStatusCode = 1

println("z=",objective_value(modelo),
 " x=", value(modelo[:x]),
 " y=", value(modelo[:y]))

z=19.25 x=1.0 y=3.25

#modelo M2
delete_upper_bound(modelo[:x])
set_lower_bound(modelo[:x], 2)
optimize!(modelo)
termination_status(modelo)

OPTIMAL::TerminationStatusCode = 1

println("z=",objective_value(modelo),
 " x=", value(modelo[:x]),
 " y=", value(modelo[:y]))

z=19.799999999999997 x=2.0 y=2.76
```

La solución óptima de la relajación de M_1 es $(x_1, y_1) = (1; 3.25)$ de valor $z_1 = 19.25$, y la solución óptima de la relajación de M_2 es $(x_2, y_2) = (2; 2.76)$ de valor $z_2 = 19.8$. Luego ambos nodos se crean activos. Más aún la cota de M_0 se actualiza a $\text{máx}\{19.25, 19.8\} = 19.8$. Por lo que el nuevo árbol de BnB (y su diagrama actual) se ven asi:



Resulta útil anotar en las aristas del árbol que restricciones hemos agregado en cada problema. Ahora hay 2 nodos activos en el árbol y podemos elegir cualquiera para ramificar. La solución (x_1, y_1) tiene variable y fraccional. Ramifiquemos M_1 en dos problemas M_3 y M_4 de acuerdo a si $y \leq 3$ o si $y \geq 4$. Obtenemos:

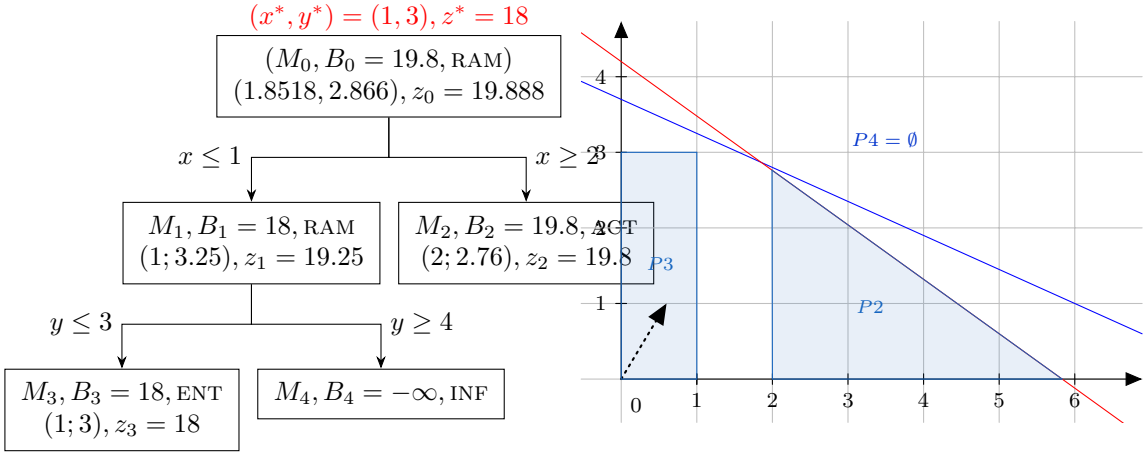
$$M_3: \quad \text{máx } 3x + 5y \qquad M_4: \quad \text{máx } 3x + 5y$$

$$S_3: \begin{cases} 20y + 9x & \leq 74 \\ 25y + 18x & \leq 105 \\ x & \leq 1 \\ y & \leq 3 \\ x, y & \geq 0 \\ x, y & \in \mathbb{Z} \end{cases} \qquad S_4: \begin{cases} 20y + 9x & \leq 74 \\ 25y + 18x & \leq 105 \\ x & \leq 1 \\ y & \geq 4 \\ x, y & \geq 0 \\ x, y & \in \mathbb{Z} \end{cases}$$

```
#Modelo M3
set_lower_bound(modelo[:x],0)
set_upper_bound(modelo[:x],1)
set_upper_bound(modelo[:y],3)
optimize!(modelo)
termination_status(modelo)

OPTIMAL::TerminationStatusCode = 1

println("z=",objective_value(modelo),
 " x=", value(modelo[:x]),
 " y=", value(modelo[:y]))

z=18.0 x=1.0 y=3.0

#Modelo M4
delete_upper_bound(modelo[:y])
set_lower_bound(modelo[:y],4)
optimize!(modelo)
termination_status(modelo)

INFEASIBLE::TerminationStatusCode
= 2
```

La solución de la relajación de M_3 es entera $(x_3, y_3) = (1, 2)$ con $z_3 = 18$. Por lo que M_3 se declara entero y además, (x_3, y_3) se transforma en incumbente (actualizando también z^*). Mientras tanto la relajación de M_4 es infactible, por lo que su cota es $-\infty$. Actualizamos la cota superior de M_1 a 18, mientras que la cota de M_0 se mantiene.

Nuestro **árbol de BnB** se ve actualmente así.



Como tenemos incumbente y cota, el **gap** de nuestra solución actual es $\frac{19.8-18}{18} = \frac{1.8}{18} = 0.1 = 10\,\%$. Solo queda M_2 activo, lo ramificamos en M_5 y M_6, donde $y \leq$ o $y \geq 3$ respectivamente.

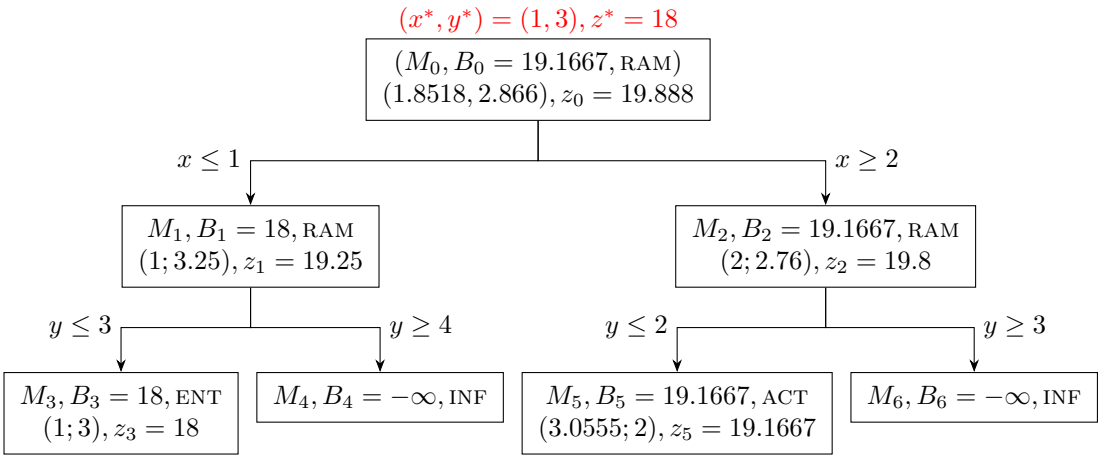
$$M_5: \quad \max 3x + 5y \qquad S_5: \begin{cases} 20y + 9x & \leq 74 \\ 25y + 18x & \leq 105 \\ x & \geq 2 \\ y & \leq 2 \\ x, y & \geq 0 \\ x, y & \in \mathbb{Z} \end{cases}$$

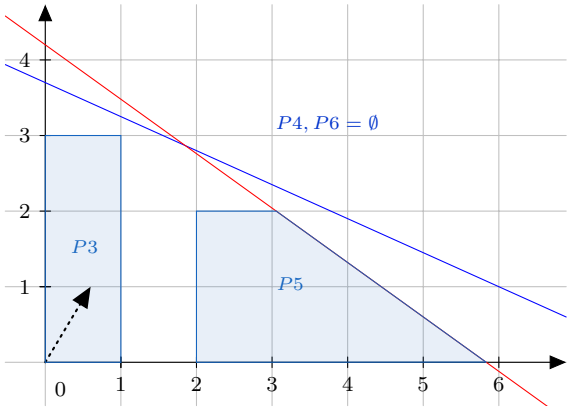
$$M_6: \quad \max 3x + 5y \qquad S_6: \begin{cases} 20y + 9x & \leq 74 \\ 25y + 18x & \leq 105 \\ x & \geq 2 \\ y & \geq 3 \\ x, y & \geq 0 \\ x, y & \in \mathbb{Z} \end{cases}$$

```
#Modelo M5
set_lower_bound(modelo[:x],2)
delete_upper_bound(modelo[:x])
set_upper_bound(modelo[:y],2)
set_lower_bound(modelo[:y],0)
optimize!(modelo)
termination_status(modelo)

OPTIMAL::TerminationStatusCode = 1

println("z=",objective_value(modelo),
 " x=", value(modelo[:x]),
 " y=", value(modelo[:y]))

z=19.166666666666664
x=3.0555555555555554 y=2.0
```

```
#Modelo M6
delete_upper_bound(modelo[:y])
set_lower_bound(modelo[:y],3)
optimize!(modelo)
termination_status(modelo)

INFEASIBLE::TerminationStatusCode
= 2
```

Notamos que M_6 es infactible, y la relajación de M_5 tiene solución $(3.0555; 2)$ con valor $z_5 = 19.1667$ por lo que queda activo (y podemos actualizar las cotas). Actualizamos el árbol BnB como sigue:





Nuestro gap mejoró a $(19.1667 - 18)/18 = 6.48\,\%$. Nuevamente tenemos solo un nodo activo, M_5.
Lo ramificamos en x, definiendo los problemas M_7 y M_8 con $x \leq 3$ y $x \geq 4$ respectivamente:

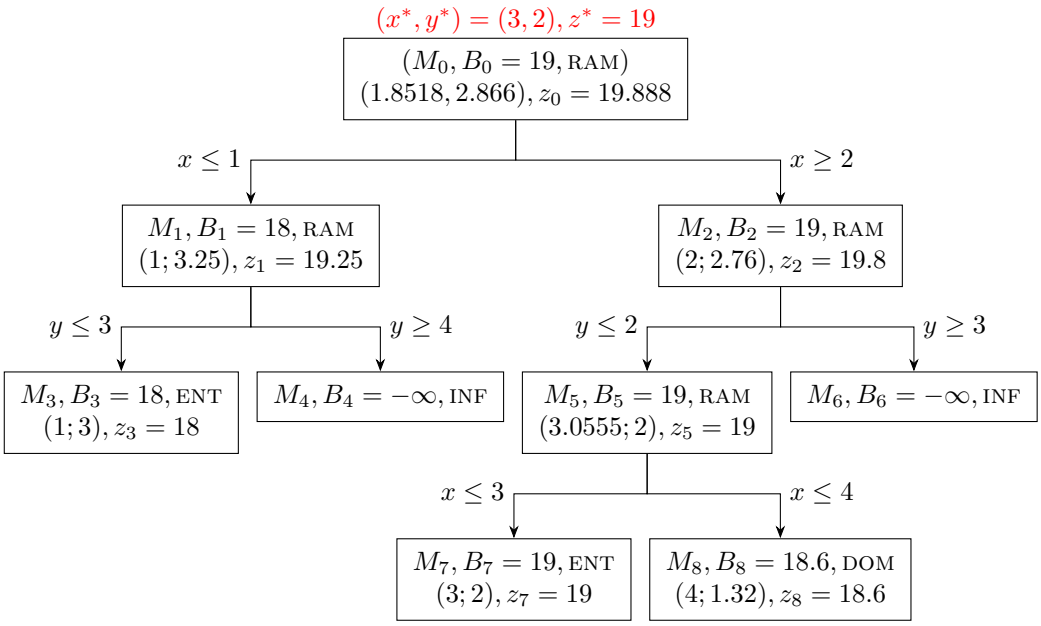
$$M_7: \quad \text{máx } 3x + 5y \qquad S_7: \begin{cases} 20y + 9x & \leq 74 \\ 25y + 18x & \leq 105 \\ x & \geq 2 \\ x & \leq 3 \\ y & \leq 2 \\ x, y & \geq 0 \\ x, y & \in \mathbb{Z} \end{cases}$$

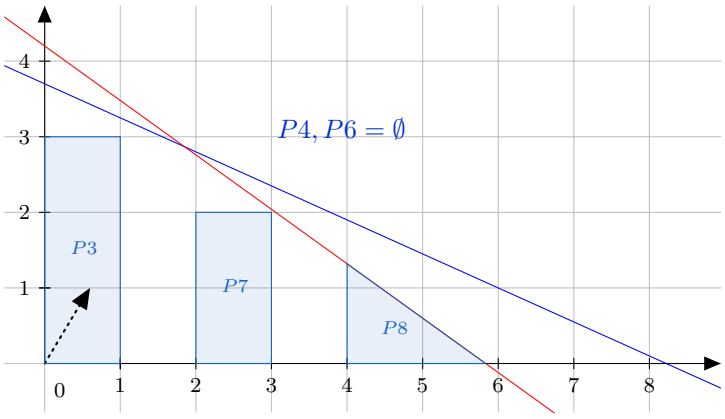
$$M_8: \quad \text{máx } 3x + 5y \qquad S_8: \begin{cases} 20y + 9x & \leq 74 \\ 25y + 18x & \leq 105 \\ x & \geq 4 \\ y & \geq 2 \\ x, y & \geq 0 \\ x, y & \in \mathbb{Z} \end{cases}$$

```
#Modelo M7
set_lower_bound(modelo[:x],0)
set_upper_bound(modelo[:x],3)
set_upper_bound(modelo[:y],2)
set_lower_bound(modelo[:y],0)
optimize!(modelo)
termination_status(modelo)

OPTIMAL::TerminationStatusCode = 1

println("z=",objective_value(modelo),
 " x=", value(modelo[:x]),
 " y=", value(modelo[:y]))

z=19.0 x=3.0 y=2.0

#Modelo M8
delete_upper_bound(modelo[:x])
set_lower_bound(modelo[:x],4)
optimize!(modelo)
termination_status(modelo)

OPTIMAL::TerminationStatusCode = 1

println("z=",objective_value(modelo),
 " x=", value(modelo[:x]),
 " y=", value(modelo[:y]))

z=18.6 x=4.0 y=1.32
```

Al resolver la relajación de M_7 se obtiene una solución **integral** $(x_7, y_7) = (3, 2)$ con $z_7 = 19$. Como 19 es mayor que nuestro valor incumbente actual, éste se actualiza. Por otro lado al resolver la relajación de M_8 se obtiene una solución fraccional $(x_8, y_8) = (4; 1.32)$ con $z_8 = 18.6$. Pero esta vez z_8 es peor que el valor incumbente (19), por lo que se declara **dominado** (o podado por cota).

Con esto hemos completado el árbol de BnB y obtenemos que la solución óptima es $(3, 2)$ de valor 19.





Hay muchos solvers comerciales que realizan BnB de manera muy eficiente (eligen buenos nodos activos para ramificar, hacen branching en buenas variables, etc.) y de hecho aplican varias heurísticas y otros algoritmos que aceleran aún más su ejecución. Por completitud, abajo anotamos una serie de instrucciones para ejecutar GUROBI sobre nuestro modelo (y dando directrices de modo que el orden de ramificado, etc. sea el mismo que nosotros usamos en el ejemplo anterior):

```
modelonuevo= Model()
@variable(modelonuevo, x>=0)
@variable(modelonuevo, y>=0)
@constraint(modelonuevo, rest1, 20y + 9x<=74)
@constraint(modelonuevo, rest2, 25y+ 18x<=105)

#función objetivo
@objective(modelonuevo, Max, 3x+5y)
```

```
#declaramos las variables como enteras
set_integer(modelonuevo[:x])
set_integer(modelonuevo[:y])

#Le indicamos a JuMP que el solver a utilizar es Gurobi y eliminamos presolver
set_optimizer(modelonuevo, Gurobi.Optimizer)
set_optimizer_attributes(modelonuevo, "Presolve" => 0, "Heuristics"=> 0, "Cuts"=> 0, "Threads" =>
1,
 "BranchDir" => -1)

#declaramos atributos (branch primero en x luego en y)
MOI.set(modelonuevo, Gurobi.VariableAttribute("BranchPriority"), x, 20)
MOI.set(modelonuevo, Gurobi.VariableAttribute("BranchPriority"), y, 1)

#resolver
optimize!(modelonuevo)
```

Esto produce:

```
Academic license - for non-commercial use only
Gurobi Optimizer version 9.0.1 build v9.0.1rc0 (win64)
Optimize a model with 2 rows, 2 columns and 4 nonzeros
Model fingerprint: 0x71254e4e
Variable types: 0 continuous, 2 integer (0 binary)
Coefficient statistics:
  Matrix range     [9e+00, 3e+01]
  Objective range  [3e+00, 5e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [7e+01, 1e+02]
Variable types: 0 continuous, 2 integer (0 binary)

Root relaxation: objective 1.988889e+01, 2 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   19.88889    0    2          -   19.88889      -     -    0s
     0     0   19.88889    0    2          -   19.88889      -     -    0s
     0     2   19.80000    0    2          -   19.80000      -     -    0s
*    1     1               1      18.0000000   19.80000  10.0%   2.0    0s
*    2     0               1      19.0000000   19.00000  0.00%   2.0    0s


Explored 3 nodes (6 simplex iterations) in 0.03 seconds
Thread count was 1 (of 8 available processors)

Solution count 2: 19 18

Optimal solution found (tolerance 1.00e-04)
Best objective 1.900000000000e+01, best bound 1.900000000000e+01, gap 0.0000%
```

Gurobi realiza un árbol de BnB parecido al nuestro, pero es más eficiente en su implementación: cada vez que un nodo se ramifica en 2 y uno de ellos es infactible o dominado, continúa bajando por el árbol (es decir no crea nuestros nodos 1, 4, 2, 6, 8) hasta que deja un par de nodos activos. Otros solvers realizan otras variantes de BnB.

2. Algunas consideraciones para hacer BnB eficiente

Resolución de relajación lineal en cada nodo. Un detalle interesante a considerar es que en general el PL (P) asociado a un nodo es muy similar al PL asociado al de su padre (son solo restricciones extras) por lo que podemos resolver (P) de manera más eficiente a partir de la solución óptima del padre y mediante **simplex dual**.

Selección de nodos activos El rendimiento de BnB depende fuertemente de la manera como se elige el nodo activo para ramificar (cuando hay varios de estos). Aquí hay dos objetivos que compiten: (1) Encontrar rápidamente un incumbente (2) Acotar rápidamente la cota B_0 del nodo raíz.

Algunas estrategias estándar para seleccionar nodos son:

1. Búsqueda en profundidad (DFS). *Ventajas*: apunta a encontrar un incumbente rápidamente con pocos nodos. *Desventajas*: puede explorar un área *mala* del árbol con nodos sin buenas soluciones.
2. Desarrollar el mejor nodo (best-node). Consiste en buscar el nodo activo (M) cuya cota (B) es la más alta posible. *Ventajas*: Las mejores soluciones integrales se deben encontrar en nodos con cotas altas, por lo que esta estrategia apunta a encontrar rápidamente buenas soluciones (o acotar rápidamente B_0) *Desventajas*: Muchos nodos deben permanecer activos por largo tiempo, provocando que se deba usar una gran cantidad de memoria.

Hay otras estrategias más avanzadas que involucran crear un estimador de cuanto debería *degradarse* el valor de la cota B en un nodo dado de acuerdo a su PL y luego elegir aquel con mayor valor estimado para la cota. En la práctica se ocupan estrategias mixtas como por ejemplo hacer DFS hasta que se encuentre un incumbente y luego seleccionar mejor nodo. Gurobi hace esto automáticamente (Cplex permite un poco más de control respecto a esto).

Selección de variable a ramificar (Ver por ejemplo Parámetro VarBranch de Gurobi

Puede que la solución fraccional $\bar{x}$ del nodo que ha sido elegido tenga múltiples variables fraccionales. Una forma estándar es elegir la variable x_j con $j \in E$ más fraccional (la más cercana a 0.5). Otra alternativa llamada *strong branching* involucra resolver pequeñas variaciones del PL original para determinar cual es la variable x_j cuya ramificación produce el mayor decrecimiento en la cota superior B del nodo a ramificar. Esta solución es más cara (involucra resolver un número de PL proporcional a las variables fraccionales) pero en la práctica resulta ser útil.

Criterios de término (Ver lista de parámetros de Gurobi)

BnB puede tomar un tiempo prohibitivo pero podemos detener el proceso en cualquier momento y, si para entonces tenemos un incumbente, podemos retornar una solución factible y una estimación (GAP relativo) de cuan cerca de ser óptima es la solución. Criterios típicos de término temprano incluyen detener la ejecución si: El tiempo de reloj excede un máximo establecido, si el número de nodos procesados excede un máximo, si la memoria necesaria excede un umbral, si el gap relativo o si el gap absoluto ($B - z^*$) es menor que una tolerancia preestablecida.

Heurísticas En muchos casos es posible determinar buenas soluciones factibles (incumbentes) en un nodo cualquiera mediante heurísticas. Por ejemplo, podemos aplicar un algoritmo simple (como glotón o programación dinámica) para encontrar una buena solución factible antes de comenzar BnB. De este modo muchas ramas pueden ser podadas rápidamente por cota al estar dominadas. Otras heurísticas típicas pueden ser usadas en cada nodo. Por ejemplo, redondear (de alguna forma inteligente) una buena solución fraccional puede producir una buena solución factible o bien, usar las soluciones enteras que podrían aparecer mientras se ejecuta SIMPLEX en un nodo. Esto se puede incorporar fácilmente al algoritmo genérico de BnB que ya vimos.

Usar presolver El presolver típicamente es capaz de reducir la complejidad del modelo de manera automática (eliminando variables, restricciones, cotas innecesarias, etc.). El presolver de Gurobi está activado por defecto.

Usar Cortes (branch and cut) En realidad BnB no es un buen método por si solo. Lo ideal es mejorar la formulación en cada nodo. Esto se puede hacer automáticamente mediante el uso de planos cortantes, como lo veremos más adelante.