Exploración de grafos

- Grafos
- Recorridos sobre grafos
  - Búsqueda primero en profundidad
  - Búsqueda primero en anchura
- Backtracking ("vuelta atrás")
  - Descripción general
  - Espacio de soluciones
  - Implementación
  - Ejemplos
- Branch & Bound ("ramificación y poda")
  - Descripción general
  - Estrategias de ramificación
  - Implementación
  - Ejemplos

Branch & Bound

"Branch and Bound" (B&B)
es una **generalización** de la técnica de backtracking:

- Se realiza un recorrido sistemático del árbol de estados de un problema, si bien ese recorrido no tiene por qué ser en profundidad, como sucedía en backtracking: usaremos una **estrategia de ramificación**.
- Además, utilizaremos **técnicas de poda** para eliminar todos aquellos nodos que no lleven a soluciones óptimas (estimando, en cada nodo, cotas del beneficio que podemos obtener a partir del mismo).

NOTA: Los algoritmos que utilizan B&B suelen ser de orden exponencial (o peor)en su peor caso.

Branch & Bound

Diferencias con backtracking

- En bactracking, tan pronto como se genera un nuevo hijo del nodo en curso, este hijo pasa a ser el nodo en curso.
- En B&B, se generan todos los hijos del nodo en curso antes de que cualquier otro nodo vivo pase a ser el nuevo nodo en curso (**no** se realiza un recorrido en profundidad).

En consecuencia:

- En backtracking, los únicos nodos vivos son los que están en el camino de la raíz al nodo en curso.
- En B&B, puede haber más nodos vivos, que se almacenan en una lista de nodos vivos.

Branch & Bound

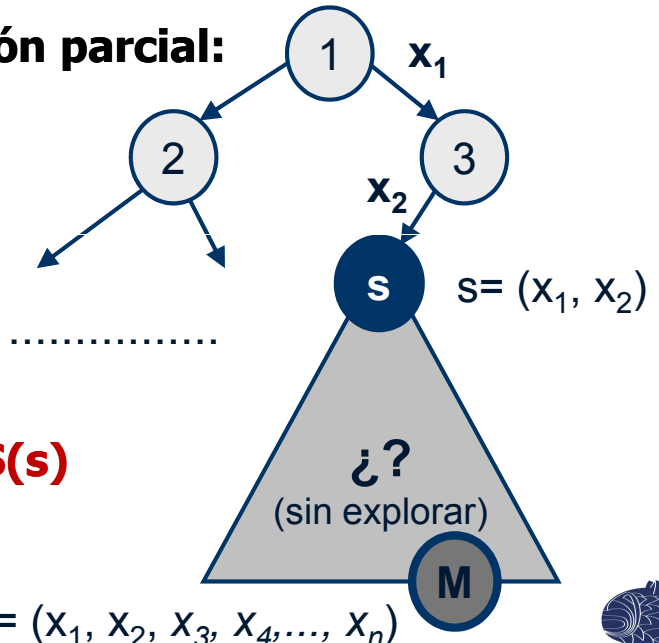
Diferencias con backtracking

- En bactracking, el test de comprobación realizado por la funciones de poda nos indica únicamente si un nodo concreto nos puede llevar a una solución o no.
- En B&B, sin embargo, se acota el valor de la solución a la que nos puede conducir un nodo concreto, de forma que esta acotación nos permite…
  - podar el árbol (si sabemos que no nos va a llevar a una solución mejor de la que ya tenemos) y
  - establecer el orden de ramificación (de modo que comenzaremos explorando las ramas más prometedores del árbol).

Branch & Bound

Estimación de las cotas
a partir de una solución parcial:

Antes de explorar **s**, se acota el valor de la mejor solución **M** alcanzable desde **s**.



$CI(s) \leq valor(M) \leq CS(s)$

Branch & Bound

Descripción general

Branch & Bound es un método general de búsqueda que se aplica de la siguiente forma:

- Explora un árbol comenzando a partir de un problema raíz y su región factible (inicialmente, el problema original, con su espacio de soluciones completo).
- Aplica funciones de acotación al problema raíz, para el que establece cotas inferiores y/o superiores.
- Si las cotas cumplen las condiciones que se hayan establecido, habremos encontrado la solución óptima del problema y la búsqueda termina.

Branch & Bound

Descripción general

- Si se encuentra una solución óptima para un subproblema concreto, ésta será una solución factible para el problema completo, pero no necesariamente su óptimo global.

- Cuando en un nodo (subproblema), su cota local es peor que el mejor valor conocido en la región, no puede existir un óptimo global en el subespacio de la región factible asociada a ese nodo y, por tanto, ese nodo puede ser eliminado ("podado").

Branch & Bound

Descripción general

En B&B, la búsqueda prosigue hasta que…

- se examinan o "podan" todos los nodos, o bien

- se cumple algún criterio pre-establecido sobre el mejor valor encontrado y las cotas locales de los subproblemas aún no resueltos.

Branch & Bound

Estimadores y cotas en Branch & Bound

	Problema de maximización	Problema de minimización
Valor	Beneficio	Coste
Cota local	Cota superior	Cota inferior
	CL ≥ Óptimo local	CL ≤ Óptimo local
	Interpretación: No alcanzaremos nada mejor al expandir el nodo.	
Cota global	Cota inferior	Cota superior
	CG ≤ Óptimo global	CG ≥ Óptimo global
	Interpretación: La solución óptima nunca será peor que esta cota.	

Branch & Bound

Estimadores y cotas en Branch & Bound:

Cota local

- Nos permite asegurar que no se alcanzará nada mejor al expandir un nodo determinado.
- Se calcula localmente para cada nodo i.
- Si ÓptimoLocal(i) es el coste/beneficio de la mejor solución que se podría alcanzar al expandir el nodo i, la cota local es una estimación de dicho valor que debe ser mejor o igual que ÓptimoLocal(i).
- Cuanto más cercana sea la cota a ÓptimoLocal(i), mejor será la cota y más se podará el árbol (si bien debemos mantener un equilibrio entre la eficiencia del cálculo de la cota y su calidad).

Branch & Bound

Estimadores y cotas en Branch & Bound:

Cota global

- La solución óptima nunca será peor que esta cota.
- Es el valor de la mejor solución estudiada hasta el momento (o una estimación del óptimo global) y debe ser peor o igual al coste/beneficio de la solución óptima.
- Inicialmente, se le puede asignar el valor obtenido por un algoritmo greedy o, en su defecto, el peor valor posible.
- Se actualiza siempre que alcanzamos una solución que mejore su valor actual.
- Cuanto más cercana sea al coste/beneficio óptimo, más se podará el árbol, por lo que es importante encontrar buenas soluciones cuanto antes.

Branch & Bound

Estimadores y cotas en Branch & Bound:

Estimador del coste/beneficio local óptimo

- Se calcula para cada nodo i y sirve para determinar el siguiente nodo que se expandirá.
- Es un estimador de ÓptimoLocal(i), como la cota local, pero no tiene por qué ser mejor o igual que ÓptimoLocal(i).
- Normalmente, se utiliza la cota local como estimador, pero, si se puede definir una medida más cercana a ÓptimoLocal(i) sin que importe si es mejor o peor que ÓptimoLocal(i), podría interesar el uso de esta medida para decidir el siguiente nodo que se expandirá.

Branch & Bound

Estrategia de poda en Branch & Bound

Además de podar aquellos nodos que no cumplan las restricciones implícitas (soluciones parciales no factibles), **se podrán podar aquellos nodos cuya cota local sea peor que la cota global**.

Si sé que lo mejor que se puede alcanzar al expandir un nodo no puede mejorar una solución que ya se ha obtenido (o se va a obtener al explorar otra rama del árbol), no es necesario expandir dicho nodo.

Branch & Bound

Estrategia de poda en Branch & Bound

Por la forma en la que están definidas las cotas local y global, se puede asegurar que con la poda no se perderá ninguna solución óptima:

Por definición:
- CotaLocal(i) es mejor o igual que ÓptimoLocal(i).
- CotaGlobal es peor o igual que Óptimo.

Si CotaLocal(i) es peor que CotaGlobal,
entonces
ÓptimoLocal(i) tiene que ser peor que Óptimo.

Branch & Bound

Estrategia de poda en Branch & Bound

	Problema de maximización	Problema de minimización
Valor	Beneficio	Coste
Podar si…	**CL < CG**	**CL > CG**
Cota local	CL ≥ Óptimo local	CL ≤ Óptimo local
	Interpretación: No alcanzaremos nada mejor al expandir el nodo.	
Cota global	CG ≤ Óptimo global	CG ≥ Óptimo global
	Interpretación: La solución óptima nunca será peor que esta cota.	

Branch & Bound

Estrategias de ramificación

- Normalmente, el árbol de estados de un problema no se representa de forma explícita.
- Para realizar el recorrido del árbol se utiliza una lista de nodos vivos (nodos generados pero aún no explorados).
- Según cómo se implemente la lista de nodos vivos, el recorrido será de uno u otro tipo.

La lista de nodos vivos contiene todos los nodos que han sido generados pero que no han sido explorados todavía (los nodos pendientes de tratar por B&B).

Branch & Bound

Estrategias de ramificación

Sin tener en cuenta los costes/beneficios, realizando una búsqueda "a ciegas":

- **Estrategia FIFO** (First In First Out)
- **Estrategia LIFO** (Last In, First Out).

Usando estimaciones de costes/beneficios, explorando primero los nodos más prometedores:

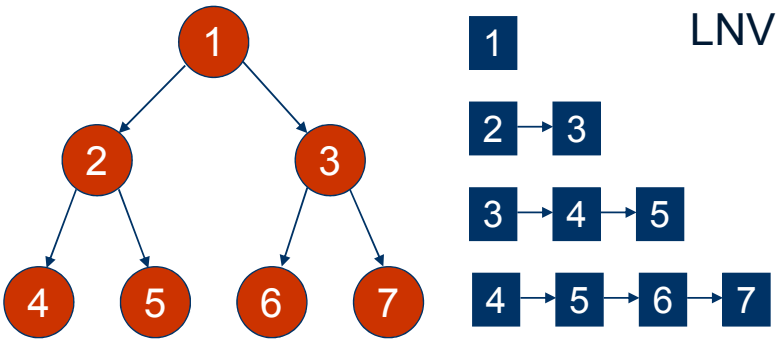
- **Estrategias LC** (Least Cost) / **MB** (Maximum Benefit).

Branch & Bound

Estrategias de ramificación: Estrategia FIFO

LNV

- Lista de nodos vivos: Cola FIFO.
- Recorrido del árbol en anchura.

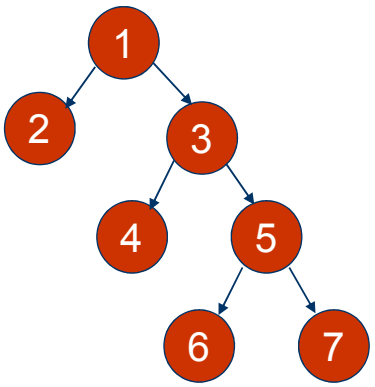


Branch & Bound

Estrategias de ramificación:
Estrategia LIFO

LNV

- Lista de nodos vivos: Pila LIFO.
- Recorrido del árbol en profundidad.



LNV

1
2 → 3
2 → 4 → 5
2 → 4 → 6 → 7

Branch & Bound

Estrategias de ramificación:
Estrategias LC [Least Cost] / MB [Maximum Benefit]

De los nodos de la lista de nodos vivos,
elegir el que tenga…

menor coste estimado (LC)
en problemas de minimización, o

mayor beneficio estimado (MB)
en problemas de maximización.

Branch & Bound

Estrategias de ramificación:
Estrategias LC [Least Cost] / MB [Maximum Benefit]

En caso de empate (de beneficio o coste estimado) deshacer el empate usando un criterio FIFO o LIFO:

- Estrategia LC-FIFO/MB-FIFO: En caso de empate, escoger el primero que se introdujo en la LNV.
- Estrategia LC-LIFO/MB-LIFO: En caso de empate, escoger el último que se introdujo en la LNV.

Branch & Bound

Estrategias de ramificación:
Estrategias LC [Least Cost] / MB [Maximum Benefit]

En cada nodo podemos tener:

- una cota inferior de coste/beneficio,
- un coste/beneficio estimado y
- una cota superior del coste/beneficio.

El árbol se poda según los valores de las cotas (CI,CS).

El árbol se ramifica según los valores estimados (E).

Branch & Bound

Branch & Bound en problemas de minimización

- La cota local es una cota inferior CI(i) del mejor coste que se puede conseguir al expandir el nodo i:

  CI(i) ≤ ÓptimoLocal(i)

- La cota global es una cota superior CS del coste del óptimo global:

  CS ≥ Óptimo

- Se puede podar un nodo i cuando **CI(i) > CS**.

Branch & Bound

Branch & Bound en problemas de maximización

- La cota local es una cota superior CS(i) del máximo beneficio que se puede conseguir al expandir el nodo i:

  CS(i) ≥ ÓptimoLocal(i)

- La cota global es una cota inferior CI del beneficio del óptimo global:

  CI ≤ Óptimo

- Se puede podar un nodo i cuando **CS(i) < CI**.

Branch & Bound

Ejemplo:
Branch & Bound usando LC-FIFO
en un problema de minimización

Criterio de poda:

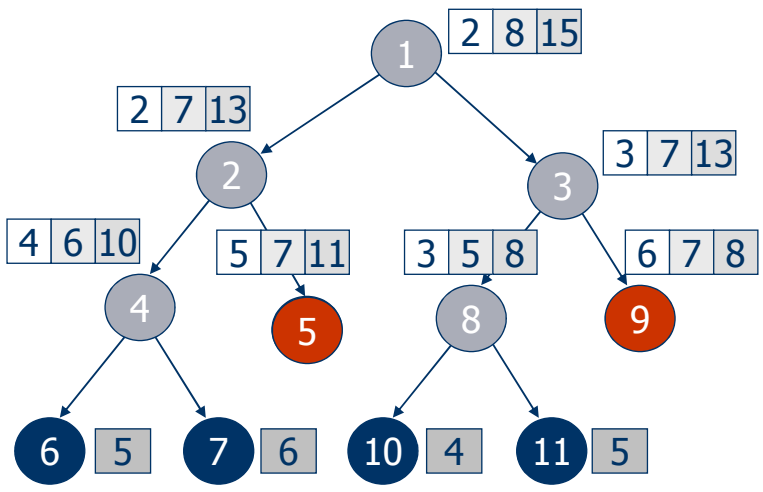
Si para algún nodo i, CI(i) > C, entonces se poda el nodo i,

donde C es valor de la menor de las cotas superiores hasta ese momento (o de alguna solución final ya encontrada).

Branch & Bound

Ejemplo:
Branch & Bound usando LC-FIFO
en un problema de minimización

CI	E	CS



C	LNV
15	1
13	2 → 3
10	4 → 3 → 5
5	3 → 5
5	8 → 5
4	5

Branch & Bound

Implementación para un problema de minimización

```
(C,s) BranchAndBoundMin (nodoRaíz)
{
  LNV    = {nodoRaíz};
  (C,s) = CS(nodoRaíz);  // Primera solución (p.ej. Greedy)
                         // y cota superior asociada

  while (LNV ≠ ∅) {

     x = seleccionar(LNV); // Según un criterio FIFO,
                           // LIFO, LC-FIFO ó LC-LIFO
     LNV = LNV - {x};
     …
```

Branch & Bound

Implementación para un problema de minimización

```
     …
     if ( CI(x) <= C )
        foreach (y hijo de x)
           if (y es una solución final mejor que s) {
              s = y;
              C = coste(y);
           } else if ( y no es solución final
                       && (CI(y) <= C) ) {
              LNV = LNV + {y};
              (Ctmp,Stmp) = CS (y);
              if (Ctmp < C) { C = Ctmp; s = Stmp; }
           }
  } // del bucle while (LNV ≠ ∅)
  return (C,s);
}
```

Branch & Bound

Implementación para un problema de maximización

```
(C,s) BranchAndBoundMax (nodoRaíz)
{
  LNV    = {nodoRaíz};
  (C,s) = CI(nodoRaíz);  // Primera solución (p.ej. Greedy)
                         // y cota inferior asociada

  while (LNV ≠ ∅) {

     x = seleccionar(LNV); // Según un criterio FIFO,
                           // LIFO, MB-FIFO ó MB-LIFO
     LNV = LNV - {x};
     …
```


Branch & Bound

Implementación para un problema de maximización

```
     …
     if ( CS(x) >= C )
        foreach (y hijo de x)
           if (y es una solución final mejor que s) {
              s = y;
              C = beneficio(y);
           } else if ( y no es solución final
                       && (CS(y) >= C) ) {
              LNV = LNV + {y};
              (Ctmp,Stmp) = CI (y);
              if (Ctmp > C) { C = Ctmp; s = Stmp; }
           }
  } // del bucle while (LNV ≠ ∅)
  return (C,s);
}
```

Branch & Bound

Observaciones:

- Sólo se comprueba el criterio de poda cuando se introduce o se saca un nodo de la lista de nodos vivos.
- Si un descendiente de un nodo es una solución final, entonces no se introduce en la lista de nodos vivos. Se comprueba si esa solución es mejor que la actual y, si es así, se actualiza C y se guarda como mejor solución hasta el momento.

Branch & Bound

¿Qué hacemos cuando CI(x) = CS(x)?

- **Opción A:** No podar (no sabemos si tenemos la solución).
- **Opción B:** Usar dos variables de poda.

  CI: Cota inferior actual de una solución parcial.
  voa: Valor óptimo actual de una solución encontrada.
  Podar **x** si (**CS(x)** < **CI**) o bien (**CS(x)** ≤ **voa**).

- **Opción C:** Generar directamente el nodo solución usando el método utilizado para calcular la cota.

```
if (CI(x) == CS(x))
   x = Solución empleada para calcular
       la cota (p.ej. algoritmo greedy)
```

Branch & Bound

Tiempo de ejecución de un algoritmo B&B

El tiempo de ejecución de un algoritmo B&B depende de:

- El número de nodos recorridos
  (que, a su vez, depende de la efectividad de la poda).
- El tiempo empleado en cada nodo
  (tiempo necesario para hacer las estimaciones de coste y gestionar la lista de nodos vivos en función de la estrategia de ramificación).

En el **peor caso**, el tiempo de un algoritmo B&B será igual al de un algoritmo backtracking (o peor incluso, si tenemos en cuenta el tiempo que requiere la LNV).

En el **caso promedio**, no obstante, se suelen obtener mejoras con respecto a backtracking.

Branch & Bound

Tiempo de ejecución de un algoritmo B&B

¿Cómo hacer que un algoritmo B&B sea más eficiente?

- Haciendo **estimaciones** de coste **muy precisas**
  (con lo que se realiza una poda exhaustiva del árbol y se recorren menos nodos, pero se emplea mucho tiempo en realizar las estimaciones).
- Haciendo **estimaciones** de coste **poco precisas**
  (con lo que se emplea poco tiempo en cada nodo, a costa de no podar demasiado, con lo que el número de nodos explorados puede ser muy elevado).

Conclusión:

Se debe buscar un **equilibrio** entre la precisión de las cotas y el tiempo empleado en calcularlas.

Branch & Bound

Soluciones branch & bound para distintos problemas

- El problema de la mochila 0/1
- El problema del viajante de comercio
- El problema de la asignación

NOTA: Para cada problema, describiremos la forma de las soluciones (y su árbol asociado), el cálculo de las cotas y las estrategias de ramificación y poda.

Branch & Bound
El problema de la mochila 0/1

1. Representación de la solución

- Mediante un **árbol binario**:
  $(s_1, s_2, \ldots, s_n)$, con $s_i \in \{0, 1\}$.

  Hijos de un nodo $(s_1,\ldots,s_k)$:
  $(s_1,\ldots,s_k,0)$ y $(s_1,\ldots,s_k,1)$.

- Mediante un **árbol combinatorio**:
  $(s_1, s_2, \ldots, s_n)$, donde $m \leq n$ y $s_i \in \{1, 2, \ldots, n\}$.

  Hijos de un nodo $(s_1,\ldots,s_k)$:
  $(s_1,\ldots,s_k,s_k+1)$, $(s_1,\ldots,s_k,s_k+2)$, …, $(s_1,\ldots,s_k,n)$.

2. Cálculo de las cotas

- **Cota inferior:** Beneficio que se obtendría sólo con los objetos incluidos hasta ese nodo.
- **Estimación del beneficio:** A la solución actual, sumar el beneficio de incluir los objetos enteros que quepan, utilizando un algoritmo greedy (p.ej. en orden decreciente de b_i/p_i).
- **Cota superior:** Valor obtenido resolviendo el problema de la mochila continuo a partir de ese nodo (un algoritmo greedy que proporciona una cota superior válida para el problema de la mochila 0/1).

Branch & Bound
El problema de la mochila 0/1

3. Estrategia de ramificación y poda

Estrategia de poda (problema de maximización):

- Variable de poda C: Valor de la mayor cota inferior o solución final del problema encontrada hasta ahora.
- Condición de poda: Podar el nodo i si **CS(i) ≤ C**.

Estrategia de ramificación:

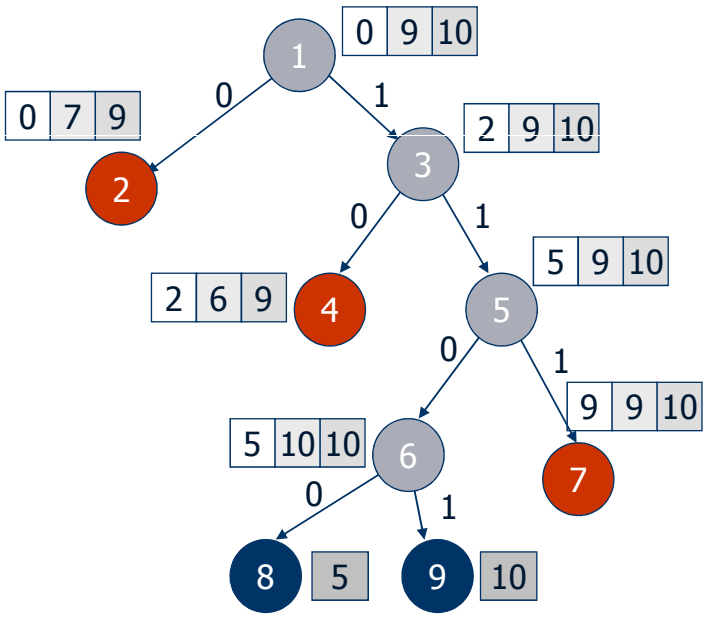
- Puesto que disponemos de una estimación del beneficio, usamos una estrategia MB (exploramos primero las ramas con mayor beneficio esperado).
- ¿MB-FIFO ó MB-LIFO? Si usásemos MB-LIFO, en caso de empate, seguiríamos por la rama más profunda.

Branch & Bound
El problema de la mochila 0/1

Ejemplo (con un árbol binario)

CI | E | CS

n = 4, M = 7, B = (2, 3, 4, 5), P = (1, 2, 3, 4)

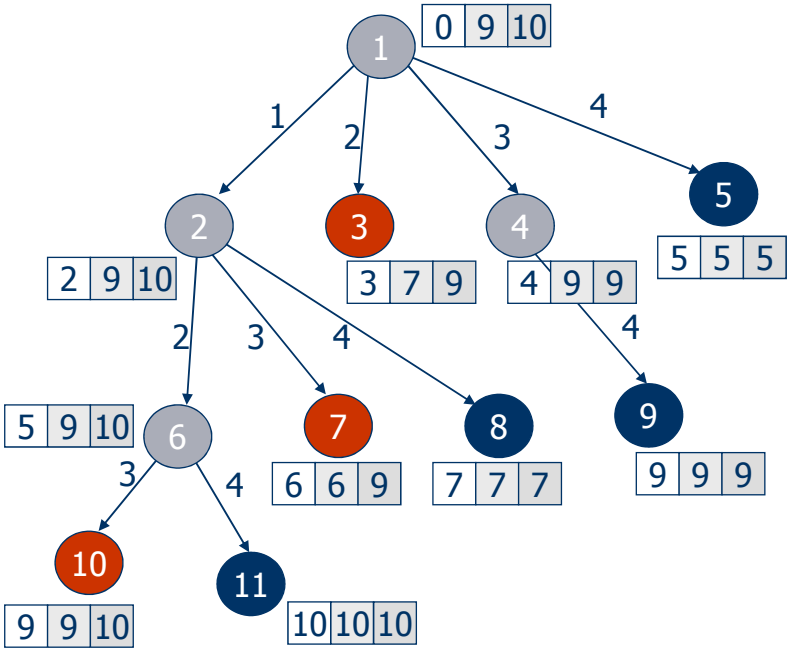


C	LNV
0	1
2	3 → 2
5	5 → 2 → 4
9	6 → 7 → 2 → 4
10	7 → 2 → 4
10	2 → 4
10	4

Branch & Bound
El problema de la mochila 0/1

Ejemplo (con un árbol combinatorio y MB-LIFO)

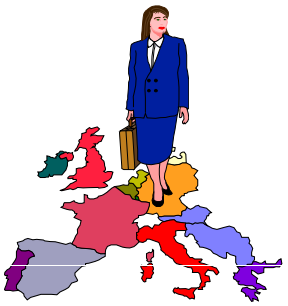
n = 4, M = 7, B = (2, 3, 4, 5), P = (1, 2, 3, 4) CI | E | CS



C	LNV
0	1
5	2 → 4 → 3
7	4 → 6 → 3 → 7
9	6 → 3 → 7
10	3 → 7
10	7

Branch & Bound
El problema del viajante de comercio

Encontrar un recorrido de longitud mínima para una persona que tiene que visitar varias ciudades y volver al punto de partida, conocida la distancia existente entre cada dos ciudades.



En términos formales:

Dado un grafo dirigido con arcos de longitud no negativa, se trata de encontrar un circuito hamiltoniano de longitud mínima (un circuito de longitud mínima que comience y termine en el mismo vértice y pase exactamente una vez por cada uno de los vértices restantes).

Branch & Bound
El problema del viajante de comercio

1. Representación del espacio de soluciones: Árbol de permutaciones restringido al grafo G

- Grafo G(V,E),
  con D[i,j] la distancia asociada a la arista $(i,j) \in E$.
- Candidatos:
  C = { (1,X,1) | X es una permutación de (2,3,…,n) },
  |C| = (n-1)!
- Soluciones factibles:
  E = { (1,X,1) | X = $(x_1,x_2,\ldots,x_{n-1})$ es una permutación de (2,3,…,n) tal que $(i_j,i_{j+1}) \in E$, $0<j<n$, $(1,x_1) \in E$, $(x_{n-1},1) \in E$}
- Función objetivo: Minimizar F(X)
  $F(X)=D[1,x_1]+D[x_1, x_2]+\ldots+D[x_{n-2}, x_{n-1}]+D[x_n,1]$

1. Representación del espacio de soluciones: Árbol de permutaciones restringido al grafo G

- La raíz del árbol (nivel 0) es el vértice inicial del ciclo.
- En el nivel i del árbol se consideran **todos** los vértices menos los i que ya han sido visitados.
- Un vértice en el nivel i debe ser adyacente a su vértice padre, que aparece en el nivel i-1 del árbol.

2. Cálculo de cotas

Cota inferior:

- El mejor coste será el de la arista adyacente que tenga el menor valor.
- La suma de los costes asociados a las mejores aristas incidentes en cada vértice aún por incluir en el circuito, más el coste del camino ya recorrido, nos ofrece una estimación válida para tomar decisiones.

2. Cálculo de cotas

Dada la siguiente matriz de adyacencia,
¿cuál es el coste mínimo de un circuito hamiltoniano?

0	14	4	10	20	⇒ Mínimo = 4
14	0	7	8	7	⇒ Mínimo = 7
4	5	0	7	16	⇒ Mínimo = 4
11	7	9	0	2	⇒ Mínimo = 2
18	7	17	4	0	⇒ Mínimo = 4
					TOTAL = **21**

3. Estrategia de ramificación y poda

Estrategia de poda (problema de minimización):

- Variable de poda C: Valor de la menor cota superior o solución final del problema encontrada hasta ahora.
- Condición de poda: Podar el nodo i si **CI(i) ≥ C**.

Estrategia de ramificación:

- Puesto que disponemos de una estimación del coste, podemos usar una **estrategia LC** (exploramos primero los circuitos con menor valor esperado).

Branch & Bound
El problema de la asignación

Enunciado del problema:

Dadas n personas y n tareas tales que B[i][j] es el rendimiento o beneficio asociado a que la persona i se encargue de realizar la tarea j, encontrar la asignación de personas a tareas que maximice el beneficio obtenido.

Formulación matemática:

Maximizar $\Sigma B[p_i][t_j]$,
sujeto a la restricción $p_i \neq p_j$, $t_i \neq t_j$, $\forall i \neq j$

Branch & Bound
El problema de la asignación

1. Representación de la solución

Desde el punto de vista de las **personas:**
$s = (t_1, t_2, ..., t_n)$, siendo $t_i \in \{1, ..., n\}$, con $t_i \neq t_j$, $\forall$ $i \neq j$

Tareas

Personas	1	2	3
1	5	6	4
2	3	8	2
3	6	5	1

2. Cálculo de cotas
Alternativa A: Estimaciones "triviales"

- **Cota inferior:**
  Beneficio acumulado hasta ese momento.
- **Cota superior:**
  CI más las restantes asignaciones con el máximo global.
- **Estimación del beneficio:**
  La media de las cotas: BE(x) = (CI(x)+CS(x))/2.

2. Cálculo de cotas
Alternativa B: Estimaciones con un algoritmo greedy

- **Cota inferior:**
  Beneficio acumulado hasta ese momento más el resultado de asignar a cada persona la tarea libre que proporciona un mayor beneficio.
- **Cota superior:**
  Asignar las tareas con mayor beneficio (aunque se repitan).
- **Estimación del beneficio:**
  La media de las cotas: BE(x) = (CI(x)+CS(x))/2.

Branch & Bound
El problema de la asignación

3. Estrategia de ramificación y poda

Estrategia de poda (problema de maximización):

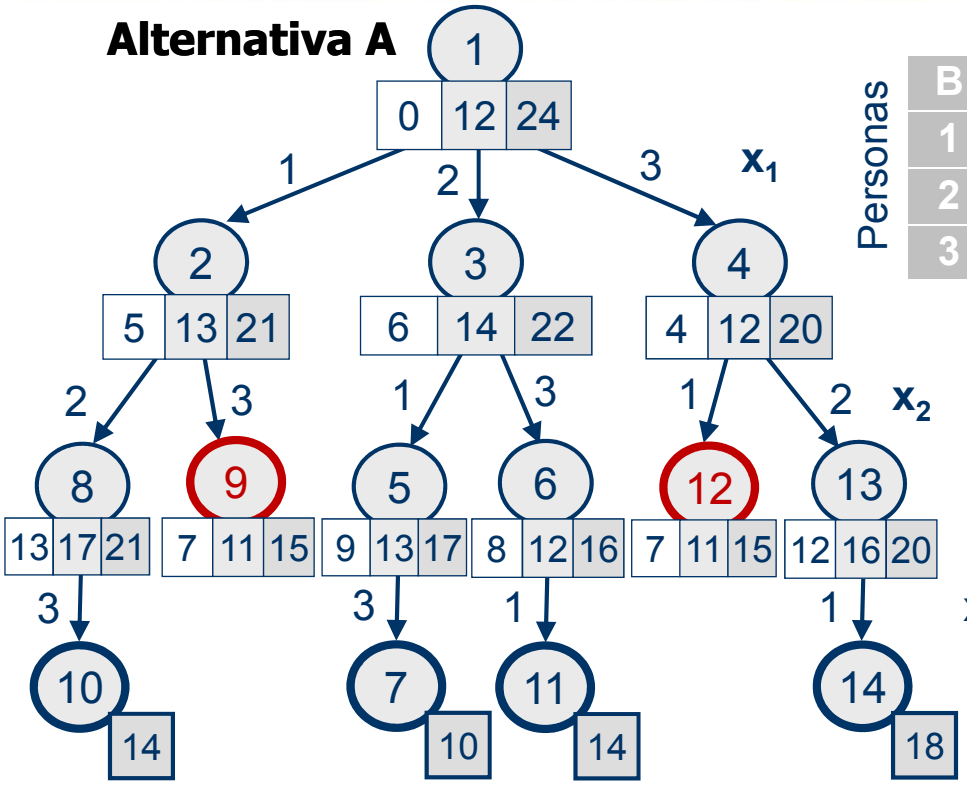
- Variable de poda C: Valor de la mayor cota inferior o solución final del problema encontrada hasta ahora.
- Condición de poda: Podar el nodo i si **CS(i) ≤ C**.

Estrategia de ramificación MB-LIFO:

- Explorar primero los nodos con mayor beneficio estimado y, en caso de empate, seguir por la rama más profunda.

Branch & Bound
El problema de la asignación

Alternativa A



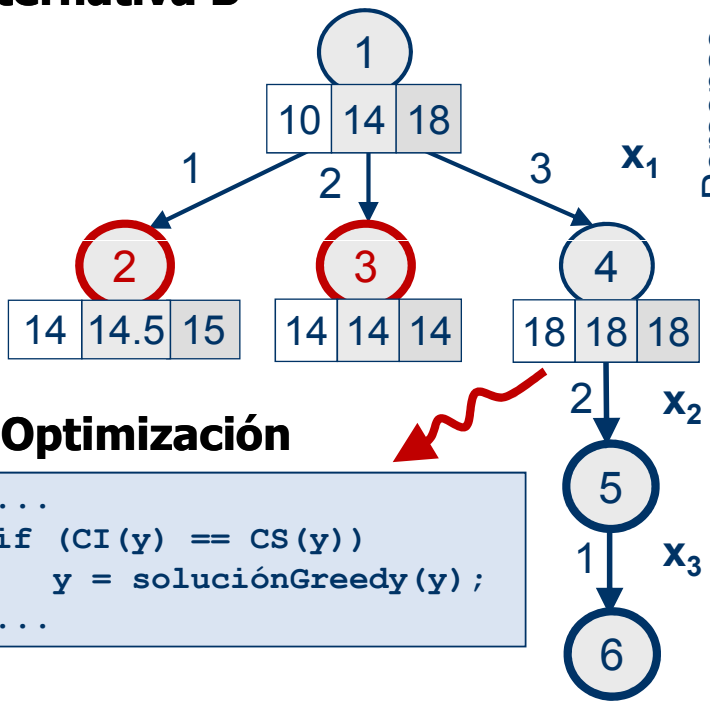
Tareas

Personas B	1	2	3
1	5	6	4
2	3	8	2
3	6	5	1

S=(3,2,1)

Branch & Bound
El problema de la asignación

Alternativa B



Tareas

Personas B	1	2	3
1	5	6	4
2	3	8	2
3	6	5	1

Optimización

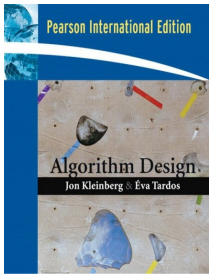
```
...
if (CI(y) == CS(y))
   y = soluciónGreedy(y);
...
```

C	LNV
10	1
18	2 3
18	3

Branch & Bound
El problema de la asignación

NOTA FINAL

El problema de la asignación se puede resolver de forma mucho más eficiente planteándolo como un problema de optimización del flujo en una red: **$O(n^3)$**.



Jon Kleinberg & Eva Tardos:
Algorithm Design, sección 7.13
Addison-Wesley, 2005
ISBN 0-321-37291-3

http://en.wikipedia.org/wiki/Assignment_problem