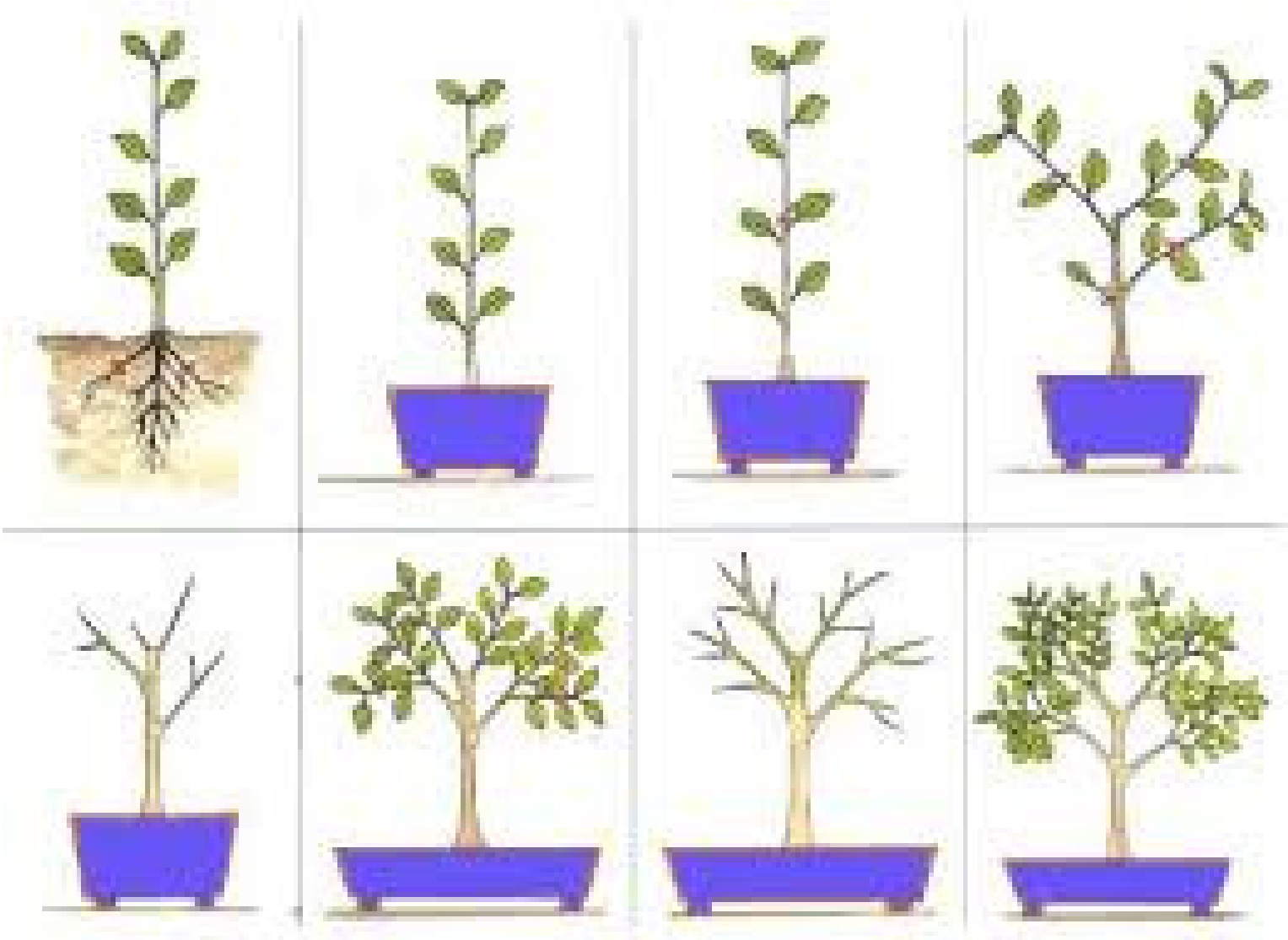
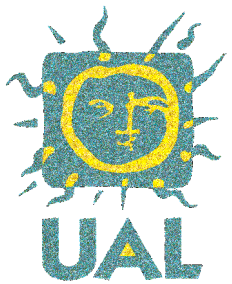
Tema 6 – Branch & Bound.

- *6.1. Introducción.*
- *6.2 Esquema general.*
- *6.3 Análisis de tiempos de ejecución.*
- *6.4 Ejemplos:*
  - *6.4.1 El problema de la Mochila 0/1.*
  - *6.4.2 El problema del viajante.*
  - *6.5.3 Planificación de tareas.*
  - *6.4.4 Otros ejemplos.*

1 – Introducción.





1 – Introducción.

Caracterización del método de "Branch & Bound":

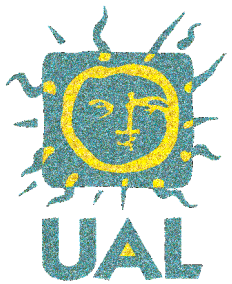
Branch and Bound es una generalización (o mejora) de la técnica de backtracking

Similitudes:

✓ Se aplica a problemas de optimización con restricciones, donde la solución se expresa como una secuencia de decisiones, y en cada decisión se elige entre un conjunto finito de valores.

✓Se genera el espacio de soluciones, organizándolo en un árbol.

✓Se realiza un recorrido sistemático del árbol de estados de un problema.



1 – Introducción.

Caracterización del método de "Branch & Bound":

Branch and Bound es una generalización (o mejora) de la técnica de backtracking

Diferencias:

✓ El recorrido del árbol de estados no tiene por qué ser en profundidad, como sucedía en backtracking; usaremos una *estrategia de ramificación*.

✓ No se genera el espacio de soluciones completo, sino que se podan bastantes estados. Se utilizan *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).

1 – Introducción.

Terminología

- Nodo *vivo*: nodo del espacio de soluciones del que no se han generado aún todos sus hijos (todavía no ha sido ramificado).

- Nodo *muerto*: nodo del que no se van a generar más hijos porque:

  - ❑ No hay más.
  - ❑ No es completable (ej: viola las restricciones).
  - ❑ No producirá una solución mejor que la solución en curso.

  Es un nodo que ya ha sido ramificado, o ha sido descartado (podado).

- Nodo *en curso* (o en expansión): nodo del que se están generando hijos (está siendo ramificado ahora). Sólo puede haber uno en cada etapa.

1 – Introducción.

Generación de hijos en “Backtracking” y “Branch & Bound”:

Backtracking:

a) Tan pronto como se genera un nuevo hijo del nodo en curso, este hijo pasa a ser el nodo en curso.

b) Los únicos nodos vivos son los que están en el camino de la raíz al nodo en curso.

c) El test de comprobación realizado por la función de poda nos indica únicamente si un nodo concreto nos puede llevar a una solución o no.

1 – Introducción.

Generación de hijos en “Backtracking” y “Branch & Bound”:

Branch & Bound:

a) 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).

b) Aparte de los que hay en el camino desde la raíz, puede haber más nodos vivos. Se deben almacenar en una estructura de datos auxiliar: *lista de nodos vivos*.

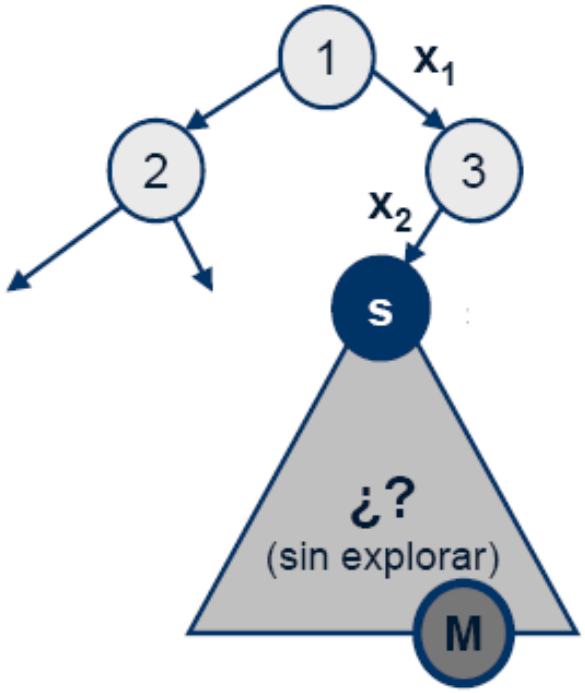
c) Se acota el valor de la solución a la que nos puede conducir un nodo concreto, de forma que esta acotación nos permite:
   - ❑ Podar el árbol (si sabemos que no nos va a llevar a una solución mejor de la que ya tenemos).
   - ❑ Establecer el orden de ramificación (comenzaremos explorando las ramas más prometedoras del árbol).

1 – Introducción.

Acotación de la solución en "Branch & Bound":

Dada una solución parcial, supongamos $s = (x_1, x_2)$, antes de explorar los descendientes del nodo correspondiente a s, se acota el valor de la mejor solución, $M = (x_1, x_2, ..., x_n)$, alcanzable desde s:

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



Si la acotación muestra que M tiene que ser necesariamente peor que la mejor solución hallada hasta el momento, no necesitamos seguir explorando por esa rama del árbol, lo que permite realizar la poda.

1 – Introducción.

Caracterización del método de “Branch & Bound”:

- La principal aportación de la técnica de branch and bound, es la posibilidad de disponer de distintas estrategias de exploración del árbol y de acotar la búsqueda de la solución, lo que se traduce en eficiencia.

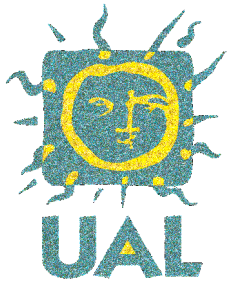
- La dificultad está en encontrar una buena función de coste/beneficio para el problema, que garantice la poda y cuyo cálculo no sea muy costoso. Si es demasiado simple, pocas ramas podrán ser desechadas. Cuanto mejor sea dicha función, mejor será el algoritmo.

1 – Introducción.

Caracterización del método de "Branch & Bound":

- Inicialmente, y antes de proceder a la poda de nodos, tendremos que disponer del coste/beneficio de la mejor solución encontrada hasta el momento, que permitirá excluir de futuras expansiones cualquier solución parcial con un coste mayor o un beneficio menor.

- Si no se desea esperar a encontrar la primera solución para empezar a podar, un buen recurso para los problemas de optimización es tomar como mejor *solución inicial* la obtenida con un algoritmo *greedy*, subóptima.



1 – Introducción.

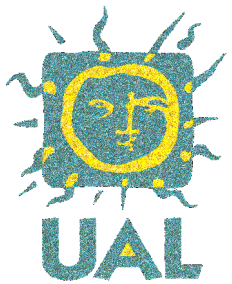
Descripción general del método de "Branch & Bound":

✓ 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.

✓ 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.



1 – Introducción.

Descripción general del método de "Branch & Bound":

✓ 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).

✓ La búsqueda prosigue hasta que se examinan o podan todos los nodos, o bien se cumple algún criterio preestablecido sobre el mejor valor encontrado y las cotas locales de los subproblemas aún no resueltos.

Elementos clave del método:

- ❑ Buen orden de recorrido.
- ❑ Función de acotación (o poda).

1 – Introducción.

Orden de recorrido del árbol en "Branch & Bound":

Un primer punto clave del método de branch and bound es encontrar *un buen orden de recorrido (o ramificación)* de los nodos, es decir, definir una buena función de prioridad de los nodos vivos, para que las soluciones buenas se encuentren rápidamente.

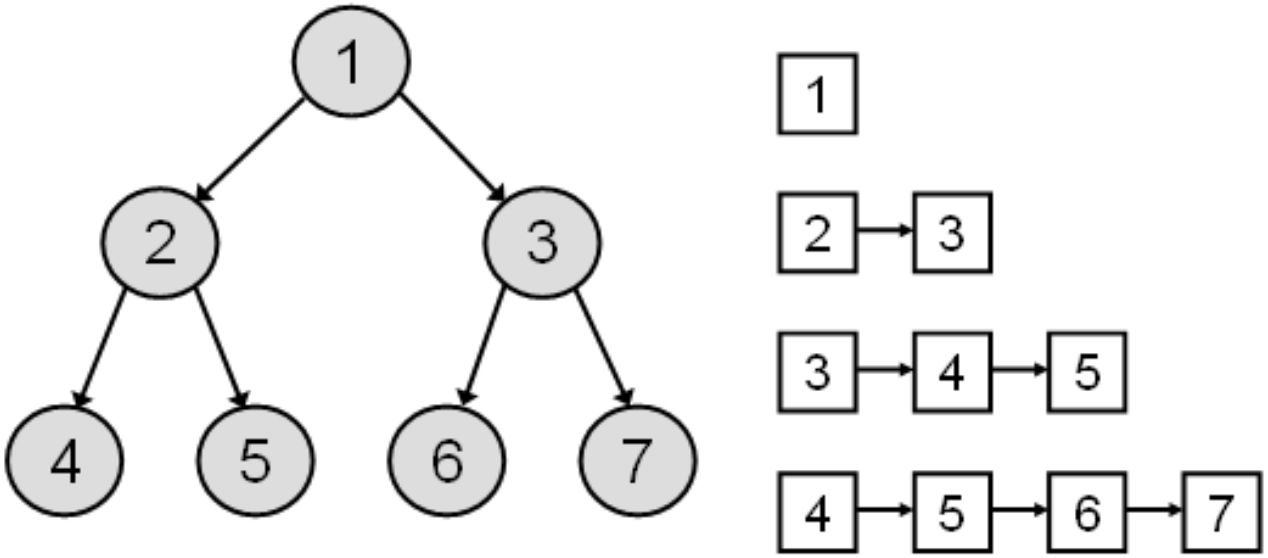
Hay distintas estrategias para *elegir el siguiente nodo de la lista de nodos vivos* (los nodos que han sido generados pero que no han sido explorados todavía, es decir, los nodos pendientes de tratar), basadas en los distintos órdenes de recorrido del árbol de soluciones:

- **FIFO**: La lista de nodos vivos es una cola, y el recorrido del árbol es por niveles (en anchura).
- **LIFO**: La lista de nodos vivos es una pila, y el recorrido del árbol es en profundidad.
- **Mínimo coste** (**LC** Least Cost) **/ Máximo beneficio** (**MB** Maximum Benefit).

1 – Introducción.

Orden de recorrido del árbol en "Branch & Bound":

- **FIFO**: La lista de nodos vivos es una cola, y el recorrido del árbol es por niveles (en anchura).

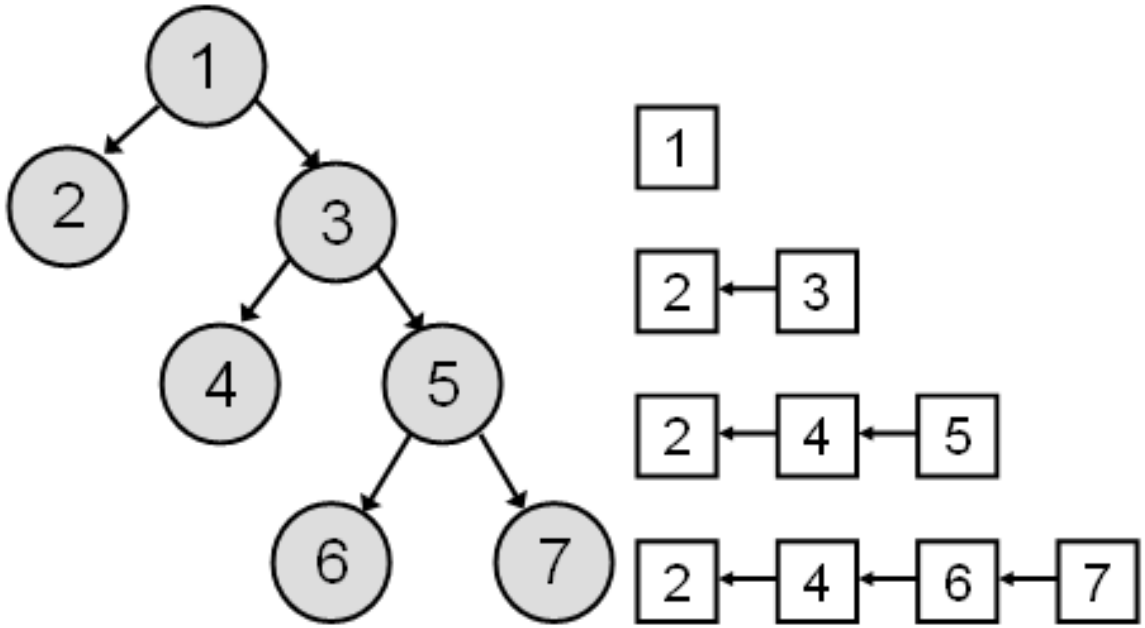


Extraer ← Lista Nodos Vivos ← **Añadir**

1 – Introducción.

Orden de recorrido del árbol en "Branch & Bound":

- **LIFO**: La lista de nodos vivos es una pila, y el recorrido del árbol es en profundidad.



Lista Nodos Vivos ← **Añadir**

→ **Extraer**

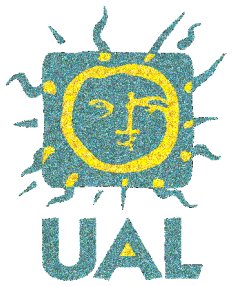
1 – Introducción.

Orden de recorrido del árbol en "Branch & Bound":

- **Mínimo coste** (**LC** Least Cost) **/ Máximo beneficio** (**MB** Maximum Benefit):

La lista es una cola de prioridad, y el recorrido es adaptado a esa situación. En este caso, la prioridad de un nodo se calcula de acuerdo con una *función de estimación de coste/beneficio*, que mide cuánto de prometedor es un nodo, explorando primero los nodos más prometedores.

En caso de empate del beneficio o coste estimado, se puede usar un criterio FIFO (escoger el primero de los empatados que se introdujo en la lista de nodos vivos) o LIFO (escoger el último que se introdujo en ella).



1 – Introducción.

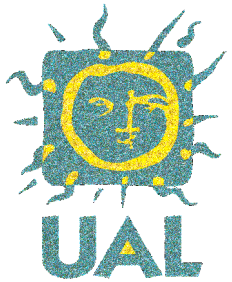
Orden de recorrido del árbol en "Branch & Bound":

- **Mínimo coste** (**LC** Least Cost) **/ Máximo beneficio** (**MB** Maximum Benefit):

En cada nodo i podemos tener:

- ✓ Una cota inferior del coste/beneficio **CI(i)**.
- ✓ Un coste/beneficio estimado **CE(i)/BE(i)**. Se puede obtener a partir de las cotas (ser la media o una de ellas).
- ✓ Una cota superior del coste/beneficio **CS(i)**.

- El árbol se **poda** según los valores de las cotas (**CI**, **CS**).
- El árbol se **ramifica** según los valores estimados (**CE/BE**), que ayudan a decidir qué parte del árbol se evaluará primero.



1 – Introducción.

Función de acotación (o poda) en “Branch & Bound”:

Los estimadores y cotas utilizados en Branch and Bound son los siguientes:

1) Cota local:

- ✓ Nos permite asegurar que no se alcanzará nada mejor al expandir un nodo determinado.
- ✓ Se calcula localmente para cada nodo i.
- ✓ Si OptimoLocal(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 OptimoLocal(i).
- ✓ Cuanto más cercana sea la cota a OptimoLocal(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).

1 – Introducción.

Función de acotación (o poda) en "Branch & Bound":

Los estimadores y cotas utilizados en Branch and Bound son los siguientes:

2) 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.

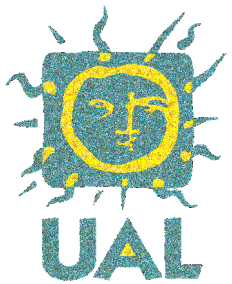
1 – Introducción.

Función de acotación (o poda) en "Branch & Bound":

Los estimadores y cotas utilizados en Branch and Bound son los siguientes:

3) 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 OptimoLocal(i), como la cota local, pero no tiene por qué ser mejor o igual que OptimoLocal(i).
- ✓ Normalmente, se utiliza la cota local como estimador, pero, si se puede definir una medida más cercana a OptimoLocal(i) sin que importe si es mejor o peor que OptimoLocal(i), podría interesar el uso de esta medida para decidir el siguiente nodo que se expandirá.



1 – Introducción.

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 sabemos 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.

- 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:
  - ✓ CotaLocal(i) es mejor o igual que OptimoLocal(i).
  - ✓ CotaGlobal es peor o igual que Óptimo.
  - ✓ Si CotaLocal(i) es peor que CotaGlobal, entonces OptimoLocal(i) tiene que ser peor que Óptimo.

1 – Introducción.

Estrategia de poda en "Branch & Bound":

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

1 – Introducción.

Estrategia de poda en "Branch & Bound":

Problemas de *maximización*:

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

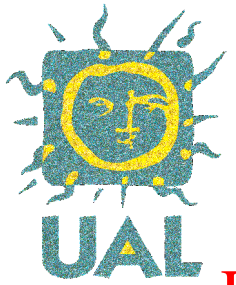
$$CS(i) \geq OptimoLocal(i)$$

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

$$CI \leq Optimo$$

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

Es decir, podar un nodo i si se cumple que $CS(i) \leq CI(j)$, para algún nodo j generado, o bien $CS(i) \leq Valor(s)$, para algún nodo s solución final.



1 – Introducción.

Estrategia de poda en "Branch & Bound":

Problemas de *minimización*:

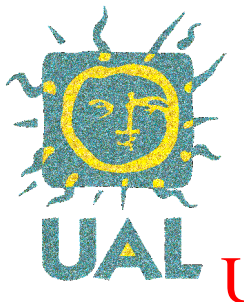
✓ La cota local es una cota inferior CI(i) del mejor coste que se puede conseguir al expandir el nodo i:

$$CI(i) \leq OptimoLocal(i)$$

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

$$CS \geq Optimo$$

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



1 – Introducción.

Ejemplo de recorrido en "Branch & Bound":

Utilizamos ramificación por mínimo coste, con estrategia FIFO para desempatar entre nodos. LC-FIFO

- Supondremos que a partir de un nodo siempre existe alguna solución.

- Criterio de poda:

  - ❑ Sea C el valor de la menor de las cotas superiores hasta ese momento (o de alguna solución final ya encontrada).

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

- Sólo se comprueba el criterio de poda cuando se introduce o se saca un nodo de la lista de nodos vivos.

1 – Introducción.

Ejemplo de recorrido en "Branch & Bound":

Utilizamos ramificación por mínimo coste, con estrategia FIFO para desempatar entre nodos. LC-FIFO

- 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 se actualizan **C** y el valor de la mejor solución óptima de forma adecuada.

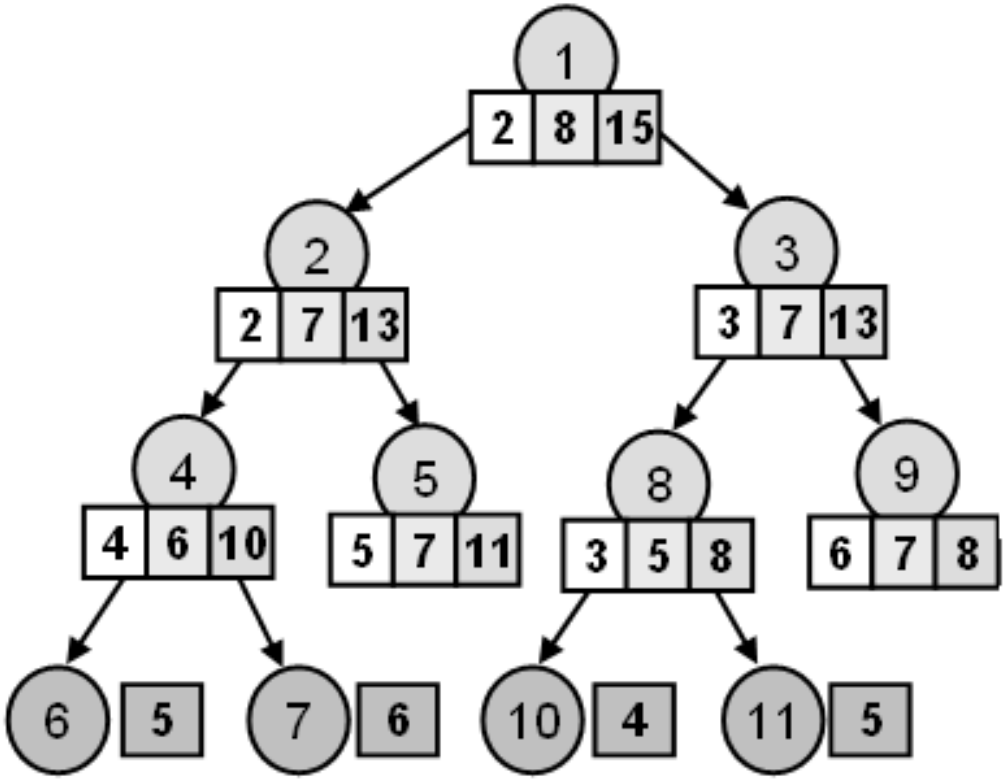
- Notación: Para cada nodo i, tendremos tres valores, con el siguiente significado:

CI (i)	CE (i)	CS (i)

+ CI(i) – Cota inferior del nodo.
+ CE(i) – Coste estimado del nodo.
+ CS(i) – Cota superior del nodo.

1 – Introducción.

Ejemplo de recorrido en "Branch & Bound":



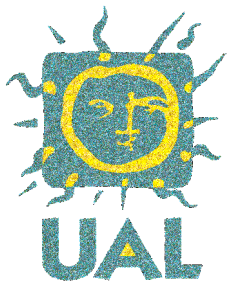
C	Lista de nodos vivos
15	1
13	2 → 3
10	4 → 3 → 5
5	3 → 5
5	8 → 5
4	5

1 – Introducción.

Ejemplo de recorrido en "Branch & Bound":

Resumen recorrido:

1. Primer nodo vivo (1) – raíz -. C (estimación global) es su cota superior.
2. Se extraen nodos hijos de (1), igual coste estimado (FIFO). Mejora estimación global C (13). Se quita (1) de la cola.
3. Se extraen nodos de (2). Se incluyen en la cola. Mejora C (10).
4. Se extraen nodos de (4). Nodo 6 es solución (nuevo valor de C), no se incluye en la cola (solución). Nodo 7 también solución, peor valor, no se considera.
5. Se extraen nodos de (3). Se incluye en la cola nodo (8), su CI < C y tiene mejor CE que (5), no se elimina y adelanta a (5) en la cola. El nodo (9) tiene un valor CI(9)>C, por lo que se poda.
6. Se extraen nodos de (9), soluciones. De las dos, la (10) da mejor resultado que la actual, nueva solución. La (11) es peor que la actual y no se considera. C pasa a ser 4 (valor de la última solución).
7. El nodo (5) se poda pues CI(5) > C actual.



2 – Esquema general.

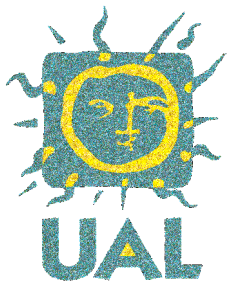
Pasos de aplicación de la técnica Branch & Bound:

1) Definir la representación de la solución. A partir de un nodo, cómo se obtienen sus descendientes.

2) Dar una manera de calcular el valor de las cotas y la estimación del beneficio.

3) Definir la estrategia de ramificación y de poda.

4) Diseñar el esquema del algoritmo.



2 – Esquema general.

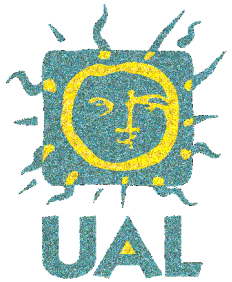
Esquema algorítmico B&B, etapas fundamentales:

Selección: Se encarga de extraer un nodo de entre el conjunto de los nodos vivos. La forma de escogerlo va a depender de la estrategia que decidamos para el algoritmo.

Ramificación: Se construyen los posibles nodos hijos del nodo seleccionado en el paso anterior.

Poda: Se eliminan algunos de los nodos creados en la etapa anterior. Esto contribuye a disminuir en lo posible el espacio de búsqueda y así disminuir la complejidad de estos algoritmos basados en la exploración de un árbol de posibilidades.

Aquellos nodos no podados pasan a formar parte del conjunto de nodos vivos, y se comienza de nuevo por el proceso de selección. El algoritmo finaliza cuando encuentra la solución, o bien cuando se agota el conjunto de nodos vivos.



2 – Esquema general.

Esquema algorítmico B&B, elementos básicos:

- Inicialización: Añadir la raíz a la lista de nodos vivos (LNV) e inicializar la variable de poda C de forma conveniente.
- Repetir mientras no se vacíe la LNV:
  - Extraer un nodo de la LNV, según la estrategia de ramificación.
  - Comprobar si debe ser podado, según la estrategia de poda.
  - En caso contrario, generar sus hijos. Para cada uno:
    - ✓ Comprobar si es una solución final y tratarla.
    - ✓ Comprobar si debe ser podado.
    - ✓ En caso contrario, añadirlo a la LNV y actualizar C de forma adecuada.

2 – Esquema general.

Esquema algorítmico B&B, para minimización de coste (LC):

```
BranchAndBoundLC (raiz: Nodo; var s: Nodo)                    // Minimización
    LNV = {raiz}
    C = CS(raiz)        // C se inicializa a la cota superior del nodo raíz.
    s = ∅               // s se podría inicializar a una solución, ej. greedy.
    mientras LNV ≠ ∅ hacer
        x = Seleccionar(LNV)        // Estrategia de ramificación
        LNV = LNV - {x}
        si CI(x) < C entonces                    // Estrategia de poda
            para cada y hijo de x hacer
                si Solución(y) AND (Valor(y)<Valor(s)) entonces
                    s = y; C = min (C, Valor(y))
                sino si NO Solución(y) AND (CI(y) < C) entonces
                    LNV = LNV + {y};  C = min (C, CS(y))
                finsi
            finpara
        finsi
    finmientras
fmétodo
```

2 – Esquema general.

Esquema algorítmico B&B, maximiza beneficio (MB):

```
BranchAndBoundMB (raiz: Nodo; var s: Nodo)                    // Maximización
    LNV = {raiz}
    C = CI(raiz)        // C se inicializa a la cota inferior del nodo raíz.
    s = ∅               // s se podría inicializar a una solución, ej. greedy.
    mientras LNV ≠ ∅ hacer
        x = Seleccionar(LNV)      // Estrategia de ramificación
        LNV = LNV - {x}
        si CS(x) > C entonces                  // Estrategia de poda
            para cada y hijo de x hacer
                si Solución(y) AND (Valor(y)>Valor(s)) entonces
                    s = y; C = max (C, Valor(y))
                sino si NO Solución(y) AND (CS(y) > C) entonces
                    LNV = LNV + {y};  C = max (C, CI(y))
                finsi
            finpara
        finsi
    finmientras
fmétodo
```

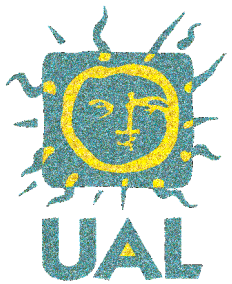
2 – Esquema general.

Esquema algorítmico B&B, (LC) y (MB):

Funciones genéricas:

- CI(i), CS(i), CE(i). Cota inferior, superior y coste estimado, respectivamente.
- Solución(x). Determina si x es una solución final válida.
- Valor(x). Valor de una solución final.
- Seleccionar(LNV): Nodo. Extrae un nodo de la LNV según la estrategia de ramificación.
- Para cada y hijo de x hacer. Iterador para generar todos los descendientes de un nodo. Equivalente a las funciones de backtracking:

```
y = x
mientras MasHermanos(nivel(x)+1, y) hacer
        Generar(nivel(x)+1, y)
        si Criterio(y) entonces ...
```



2 – Esquema general.

Esquema algorítmico B&B, (LC) y (MB), comentarios:

- Sólo se comprueba el criterio de poda cuando se introduce o se extrae 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.

2 – Esquema general.

Esquema algorítmico B&B, (LC) y (MB), caso especial:

Caso especial: CI(x) = CS(x).

+ No podar (no sabemos si tenemos la solución).

+ 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).

+ Generar directamente el nodo solución usando el método utilizado para calcular la cota:

  si (CI(x) = CS(x))

  x = Solución empleada para calcular la cota

  (ej: algoritmo greedy)

3 – Análisis de tiempos de ejecución.

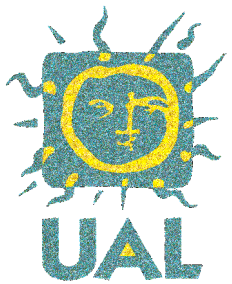
Consideraciones generales:

- La técnica branch and bound da lugar a algoritmos de complejidad exponencial, por lo que normalmente se utiliza en problemas complejos que no pueden resolverse en tiempo polinómico (NP-completos).

- El tiempo de ejecución de un algoritmo "Branch & Bound" 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).



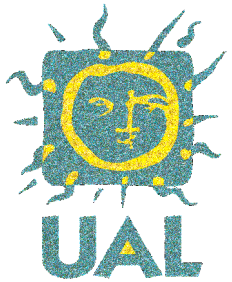
3 – Análisis de tiempos de ejecución.

Casos peor y promedio:

En el **peor caso**, el tiempo de un algoritmo "Branch & Bound" será igual al de un algoritmo backtracking (o peor incluso, si tenemos en cuenta el tiempo que requiere el cálculo de cotas y la gestión de la lista de nodos vivos).

Los algoritmos que utilizan "Branch & Bound" suelen ser de orden exponencial (o peor) en su peor caso.

En el **caso promedio**, no obstante, se suelen obtener mejoras con respecto a backtracking, dependiendo de las funciones de poda.

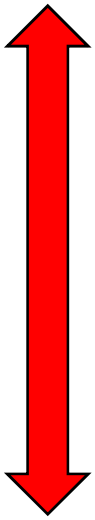


3 – Análisis de tiempos de ejecución.

Mejoras de eficiencia de un algoritmo Branch & Bound:

✓ Haciendo *estimaciones de coste y cotas 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 y cotas 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).



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

3 – Análisis de tiempos de ejecución.

Eficiencia en memoria de Branch & Bound:

La *requerimientos de memoria*, son mayores que los de los algoritmos con backtracking.

- Ya no se puede disponer de una estructura global donde ir construyendo la solución, puesto que el proceso de construcción no es tan sistemático como antes. Anotación de los recorridos más compleja.
- Ahora, cada nodo debe ser autónomo, en el sentido de contener toda la información necesaria para la ramificación y la poda, y para reconstruir la solución encontrada hasta ese momento. Aumenta el costo en memoria de cada nodo.
- Los nodos vivos pueden ser más de n, siendo n la profundidad del árbol (número mayor que con Bactracking). Aumenta el número de nodos.

4 – Ejemplos: El problema de la Mochila 0/1.

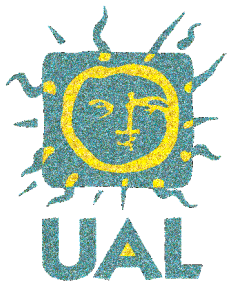
Planteamiento del problema:

+ Datos: n: número de objetos disponibles.
M: capacidad de la mochila.
$p = (p_1, p_2, ..., p_n)$ pesos de los objetos.
$b = (b_1, b_2, ..., b_n)$ beneficios de los objetos.

+ Solución: $\{x_1,..,x_i,...,x_n\} / x_i \in \{0,1\}$.

+ Objetivo: Seleccionar los objetos que nos garantizan un beneficio máximo, pero con un peso global menor o igual que M.

Maximizar $(\Sigma\, x_i \cdot b_i) / \Sigma\, x_i \cdot p_i \leq M$ con $i=1..n$, $x_i \in \{0, 1\}$.



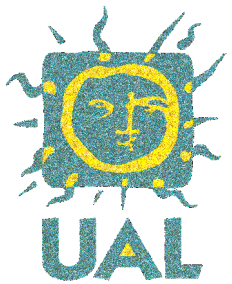
4 – Ejemplos: El problema de la Mochila 0/1.

Diseño del algoritmo branch and bound:

1. Definir una representación de la solución: a partir de un nodo, cómo se obtienen sus descendientes.

2. Indicar cómo calcular el valor de las cotas y la estimación del beneficio.

3. Definir la estrategia de ramificación y de poda.



4 – Ejemplos: El problema de la Mochila 0/1.

Representación de la solución:

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

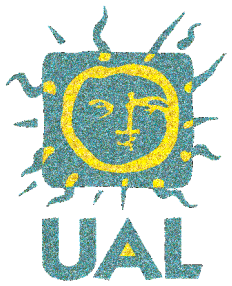
- $s_i = 0 \rightarrow$ No se coge el objeto i; $s_i = 1 \rightarrow$ Sí se coge i.
- Hijos de un nodo $(s_1, s_2, ..., s_k)$: $(s_1, ..., s_k, 0)$ y $(s_1, ..., s_k, 1)$.

Con un **árbol combinatorio**:

$(s_1, s_2, ..., s_m)$, con $m \leq n$ y $s_i \in \{1, 2, ..., n\}$.

- Hijos de un nodo $(s_1, ..., s_k)$:

$(s_1, ..., s_k, s_k + 1)$, $(s_1, ..., s_k, s_k + 2)$, ..., $(s_1, ..., s_k, n)$.



4 – Ejemplos: El problema de la Mochila 0/1.

Cálculo de cotas:

Cota inferior: Beneficio que se obtendría incluyendo sólo los objetos guardados hasta ese nodo.

Estimación del beneficio: A la solución actual, sumar el beneficio de incluir los objetos enteros que quepan, utilizando la técnica greedy. Suponemos que los objetos están ordenados por orden decreciente de b_i/p_i (preprocesamiento).

Cota superior: Resolver el problema de la mochila no 0/1 a partir de ese nodo (con un algoritmo voraz), y quedarse con la parte entera.

4 – Ejemplos: El problema de la Mochila 0/1.

Ejemplo:

$n = 4, M = 7, b = (2, 3, 4, 5), p = (1, 2, 3, 4)$

	árbol binario	árbol combinatorio
Nodo actual:	(1, 1)	(1, 2)
Hijos:	(1, 1, 0), (1, 1, 1)	(1, 2, 3), (1, 2, 4)

- Cota inferior: $CI = b_1 + b_2 = 2 + 3 = 5$
- Estimación del beneficio: $EB = CI + b_3 = 5 + 4 = 9$
- Cota superior: $CS = CI + \lfloor \text{MochilaVoraz}(3, 4) \rfloor = 5 + \lfloor 4 + 5/4 \rfloor = 10$

4 – Ejemplos: El problema de la Mochila 0/1.

Estrategia de poda:

- En una variable C guardar el valor de la mayor cota inferior hasta ese momento.

- Si para un nodo su cota superior es menor que C (a veces se usa menor o igual), se puede podar ese nodo.

4 – Ejemplos: El problema de la Mochila 0/1.

Estrategia de ramificación:

- Puesto que tenemos una estimación del beneficio, usar una estrategia MB: explorar primero las ramas con mayor valor esperado (EB).

- En caso de empate, seguir por la rama más profunda: MB-LIFO.

4 – Ejemplos: El problema de la Mochila 0/1.

Esquema algorítmico:

atributos clase nodo

b_act, p_act: entero

CI, BE, CS: entero

tupla: array [1..n] de entero

nivel: entero

4 – Ejemplos: El problema de la Mochila 0/1.

Esquema algorítmico:

```
nodo Mochila01B&B (int b[n], p[n], M)
        inic=NodoInicial(b, p, M); C=inic.CI; LNV={inic}; s.b_act = -∞
        Mientras LNV ≠ ∅ hacer
           x = Seleccionar (LNV)  { Según el criterio MB-LIFO }
           LNV = LNV - {x}
           Si  (x.CS > C)   { Si no se cumple se poda x }
                 Para  i = 0, 1
                       y = Generar (x, i, b, p, M)
                       Si (y.nivel = n)  Y  (y.b_act > s.b_act)
                             s = y; C = max {C, s.b_act}
                       si no
                            Si (y.nivel < n)  Y  (y.CS > C)
                                 LNV=LNV+{y}; C=max{C, y.CI}  fin_si
                       fin_si
                 fin_para
           fin_si
        fin_mientras
        return s
```

4 – Ejemplos: El problema de la Mochila 0/1.

Esquema algorítmico:

```
nodo NodoInicial (int b[n], p[n], M)

        res.CI = 0 ; res.CS = ⌊MochilaVoraz (1, M, b, p)⌋
        res.BE = Mochila01Voraz (1, M, b, p)
        res.nivel = 0 ; res.b_act = 0 ; res.p_act = 0 ; res.tupla ← 0
        return res
fmétodo
```

4 – Ejemplos: El problema de la Mochila 0/1.

Esquema algorítmico:

```
nodo Generar (nodo x; entero i (0, 1); entero b[n], p[n], M)
    res.tupla = x.tupla ; res.nivel = x.nivel + 1; res.tupla[res.nivel] = i
    si  (i = 0)
            res.b_act = x.b_act;  res.p_act = x.p_act
    sino
            res.b_act = x.b_act + b[res.nivel]
            res.p_act = x.p_act + p[res.nivel]
    fin_si
    res.CI = res.b_act
    res.BE = res.CI + Mochila01Voraz (res.nivel+1, M - res.p_act, b, p)
    res.CS = res.CI + ⌊MochilaVoraz (res.nivel+1, M - res.p_act, b, p)⌋
    Si res.p_act > M        { Sobrepasa el peso M: descartar el nodo }
       res.CI = res.CS = res.BE = -∞
    fin_si
    return res
fmétodo
```

4 – Ejemplos: El problema de la Mochila 0/1.

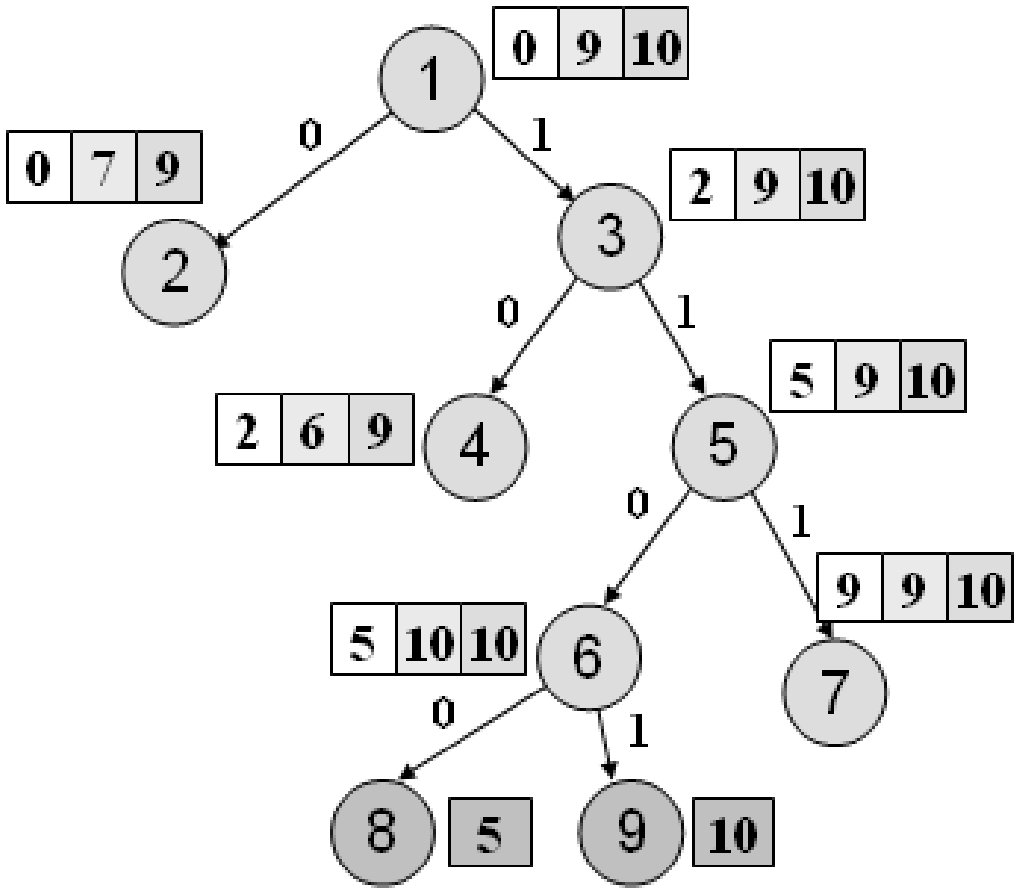
Volvamos a nuestro ejemplo:

n = 4, M = 7, b = (2, 3, 4, 5), p = (1, 2, 3, 4).

- $CI_i = \sum b_i$ incluidos hasta el momento
- $EB_i = EB_i + \sum b_j$ restantes que quepan enteros (greedy)
- $CS_i = CI_i + \lfloor$solución Mochila fraccionaria greedy a partir de ahí$\rfloor$
- C = máxima CI hasta el momento o máxima solución encontrada
- Si $CS_i \leq C$ entonces podar nodo i

4 – Ejemplos: El problema de la Mochila 0/1.

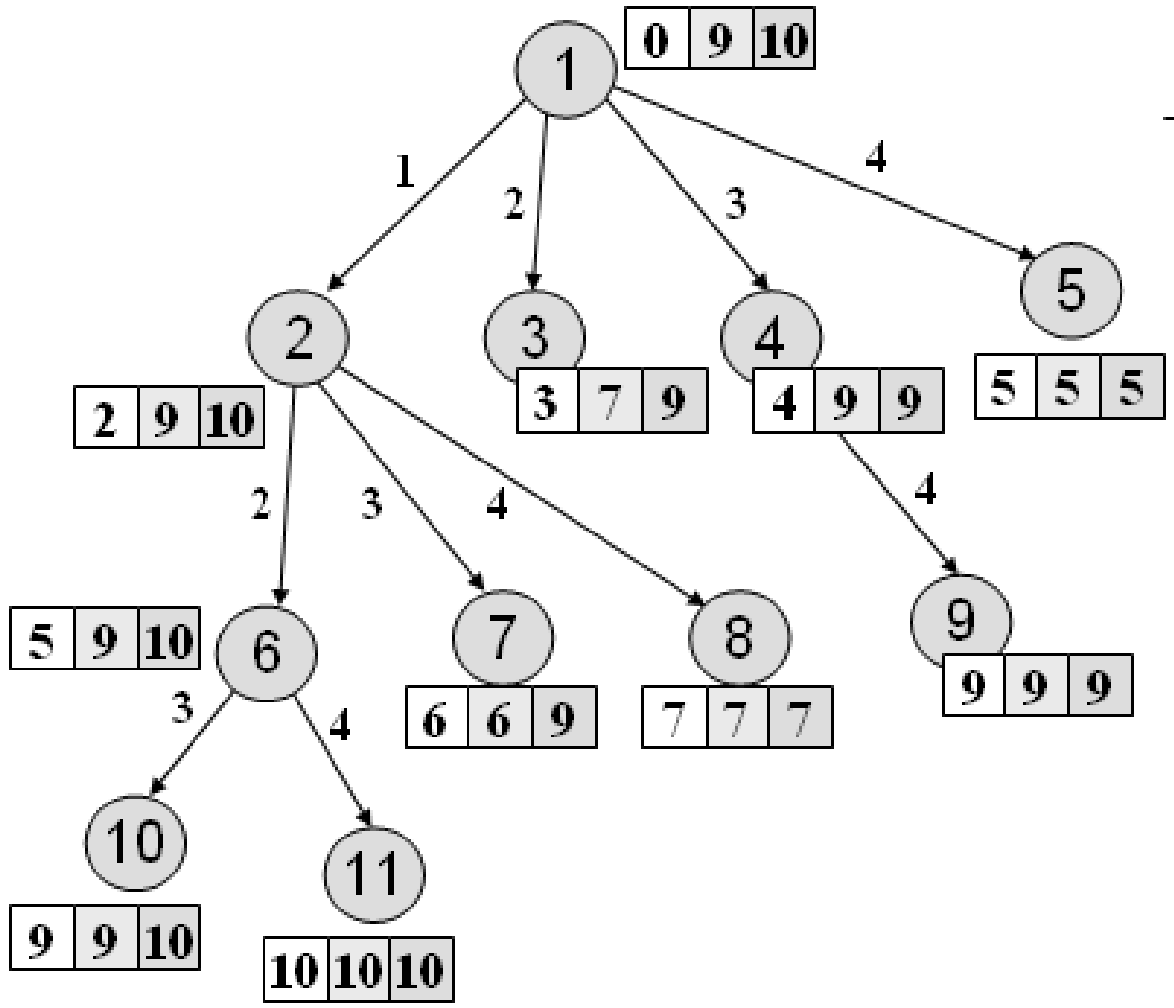
Utilizando un árbol binario y MB-LIFO:



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

4 – Ejemplos: El problema de la Mochila 0/1.

Utilizando esta vez un árbol combinatorio y MB-FIFO:



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

4 – Ejemplos: El problema del viajante.

Formulación del problema:

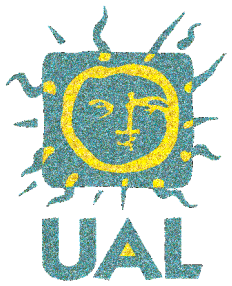
- Encontrar un recorrido de longitud mínima para una persona que tiene que visitar varias ciudades y volver al punto de partida, conocida la distancia existente entre cada dos ciudades.

- En términos formales: Dado un grafo dirigido con arcos de longitud no negativa, 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).

4 – Ejemplos: El problema del viajante.

Soluciones estudiadas hasta ahora:

- Heurística voraz: calcula una solución subóptima.
- Programación dinámica: coste en tiempo O ($n^2 \cdot 2^n$).
- Existen muchas más en la bibliografía...



4 – Ejemplos: El problema del viajante.

Representación del espacio de soluciones como árbol de permutaciones restringido al grafo G:

- Grafo G(V, A), con D[i, j] distancia asociada a la arista $(i, j) \in A$.

- El circuito buscado empieza en el vértice 1.

- 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 $(i_j, i_{j+1}) \in A$, $0 < j < n$, $(1, x_1) \in A$, $(x_{n-1}, 1) \in A$ }

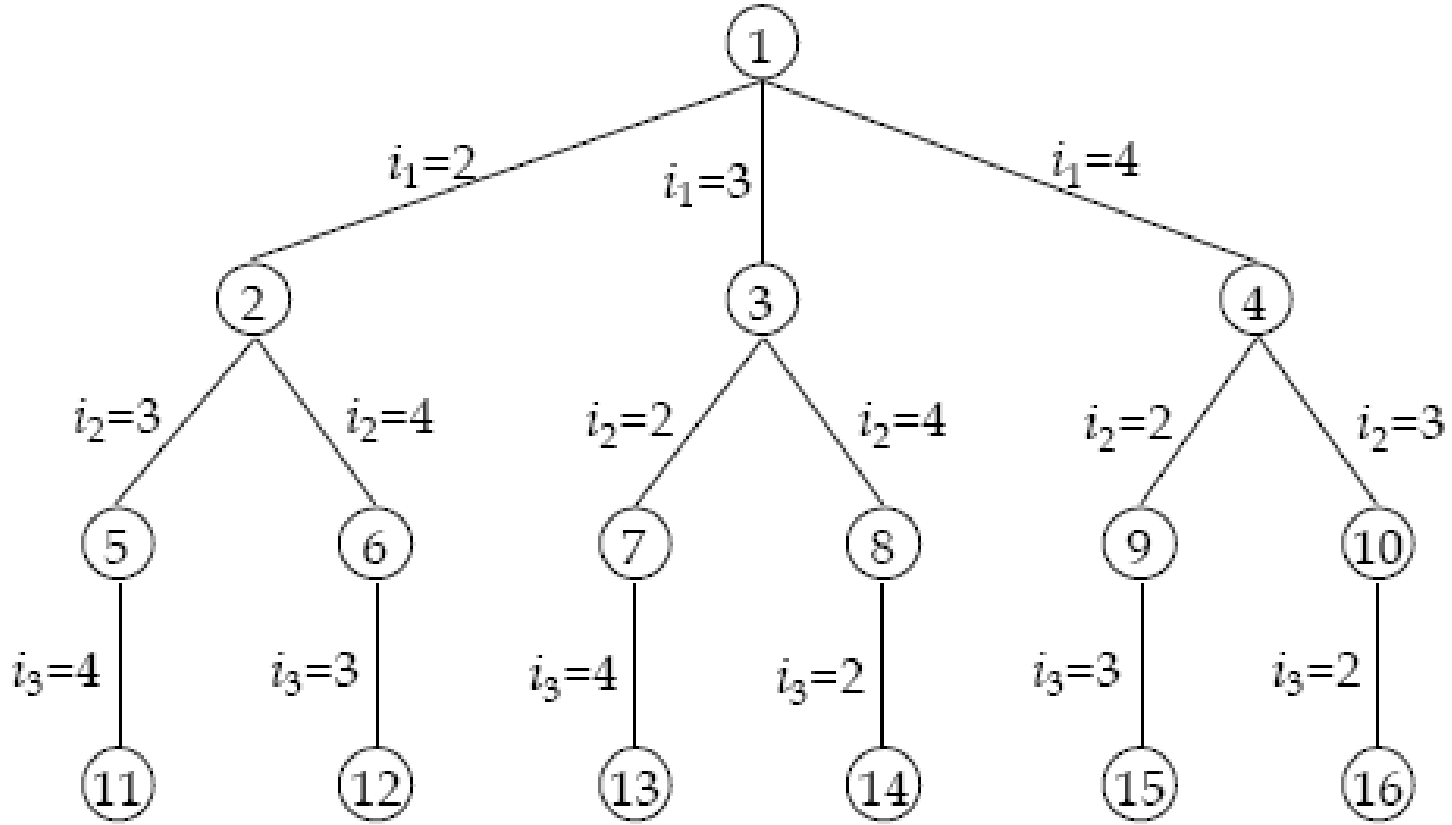
4 – Ejemplos: El problema del viajante.

Representación del espacio de soluciones como árbol de permutaciones restringido al grafo G:

- Función objetivo: Minimizar $F(X)=D[1, x_1]+D[x_1, x_2]+...+D[x_{n-2}, x_{n-1}]+D[x_{n-1}, 1]$
- 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.

4 – Ejemplos: El problema del viajante.

Ejemplo de representación del espacio de soluciones, para el caso de un grafo completo con |V| = 4:



Cada hoja es una solución y representa el viaje definido por el camino desde la raíz hasta la hoja. Hay que tener en cuenta que, para completar el circuito, hay que regresar al punto de partida.

4 – Ejemplos: El problema del viajante.

Cálculo de cotas (problema de *minimización*):

- Aunque existen otras mejores, la elección *más simple* para la **cota inferior** de cada nodo, CI(i), es la distancia total (suma de los valores de las aristas) del recorrido definido por el camino desde la raíz hasta i.

- Podemos utilizar este mismo valor como coste estimado para cada nodo, EC(i), a la hora de situar el nodo en la cola de prioridad de la LNV.

- Si el nodo i es un nodo hoja, entonces el coste de una solución vendrá dado por el valor de la estimación anterior, más el valor de la arista de regreso al vértice de partida del circuito.

- Como **cota superior**, por simplicidad, usaremos la función de acotación trivial, es decir, CS(i) = ∞.

4 – Ejemplos: El problema del viajante.

Estrategia de poda (problema de *minimización*):

- Variable de poda C: Valor de la menor cota superior o solución final del problema encontrada hasta el momento. Se comienza con C = ∞, y se cambia ese valor cuando se llega a un nodo hoja, cuyo valor de la solución sea menor.

- Criterio de poda: Podar el nodo i si CI(i) ≥ C.

4 – Ejemplos: El problema del viajante.

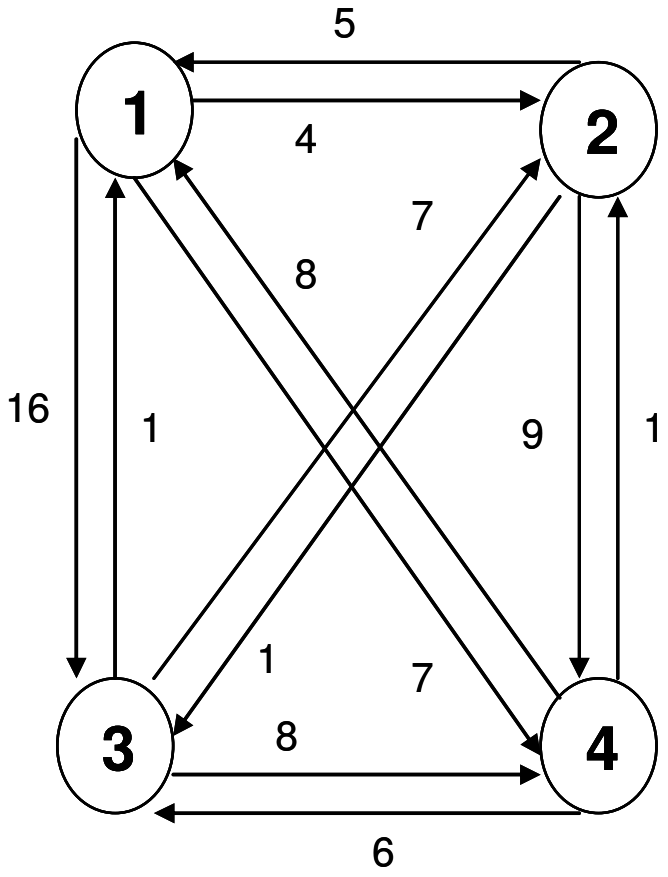
Estrategia de ramificación:

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

- Para deshacer empates, podemos usar la estrategia LC-LIFO, que explora primero la rama más profunda (más próxima a una solución).

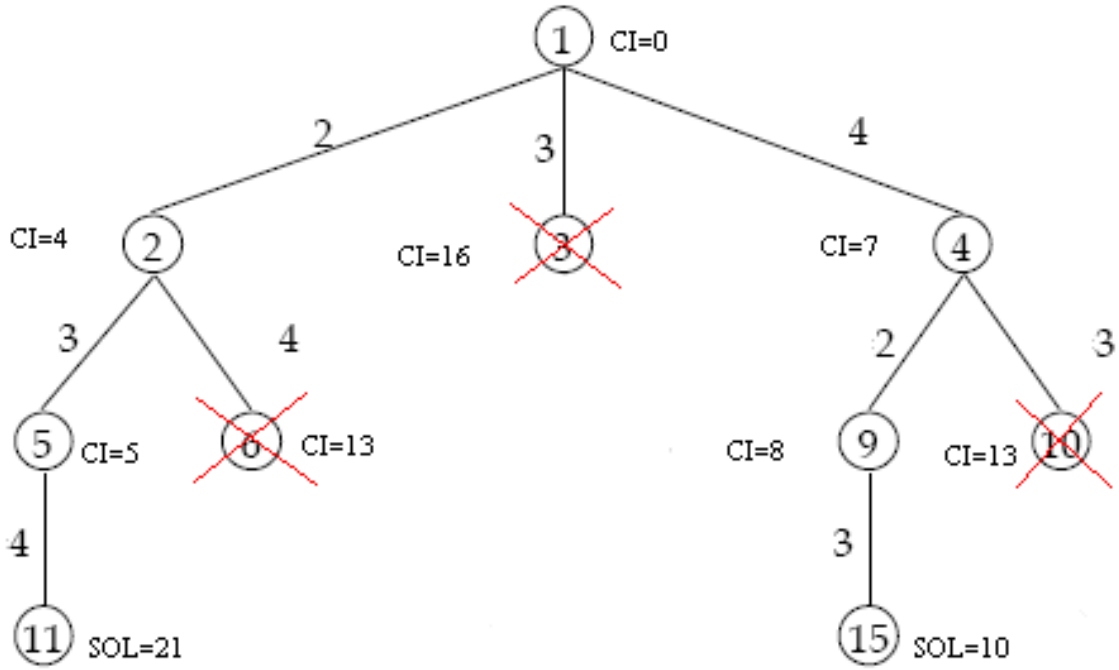
4 – Ejemplos: El problema del viajante.

Ejemplo con un grafo completo con 4 nodos:



4 – Ejemplos: El problema del viajante.

Ejemplo con un grafo completo con 4 nodos:



C	LNV
∞	1(0)
	Se insertan sus hijos:
∞	2(4) 4(7) 3(16)
	Se insertan los hijos del 2:
∞	5(5) 4(7) 6(13) 3(16)
	El 5 genera solución, con nuevo valor de C:
21	4(7) 6(13) 3(16)
	Se insertan los hijos del 4, con criterio LIFO:
21	9(8) 10(13) 6(13) 3(16)
	El 9 genera solución, con nuevo valor de C:
10	10(13) 6(13) 3(16)
	El 10 se poda por superar C:
10	6(13) 3(16)
	El 6 se poda por superar C:
10	3(16)
	El 3 se poda por superar C:
10	∅

4 – Ejemplos: El problema del viajante.

Ejemplo con un grafo completo con 4 nodos:

- Obsérvese que, en este ejemplo (no tiene por qué ser siempre así), la primera solución que hemos encontrado coincide con la del enfoque greedy:

  Circuito = 1, 2, 3, 4, 1

  Coste = 4+1+8+8=21 (incluimos la arista de retorno al vértice de comienzo).

- La solución encontrada por branch and bound, que es la solución óptima, es:

  Circuito = 1, 4, 2, 3, 1

  Coste = 7+1+1+1=10 (incluimos la arista de retorno al vértice de comienzo).

4 – Ejemplos: Planificación de tareas.

Planteamiento del problema: (igual que en Backtracking)

Vamos a considerar la siguiente situación:

- ✓ Existen n personas y n trabajos.
- ✓ Cada persona i puede realizar un trabajo j con más o menos rendimiento: B[i, j].
- ✓ Objetivo: asignar una tarea a cada trabajador (asignación uno-a-uno), de manera que se maximice la suma de rendimientos.

	Tareas			
Personas	**B**	**1**	**2**	**3**
	1	**4**	**9**	**1**
	2	**7**	**2**	**3**
	3	**6**	**3**	**5**

Problema NP Completo

Ejemplo 1 : (P1, T1), (P2, T3), (P3, T2) B_{TOTAL}= 4+3+3 = 10

Ejemplo 2 : (P1, T2), (P2, T1), (P3, T3) B_{TOTAL}= 9+7+5 = 21

4 – Ejemplos: Planificación de tareas.

Planteamiento de la solución: representación (uno de mejores que se vieron con Backtracking)

- Vector de asignaciones desde el punto de vista de las personas:

  $s = (t_1, t_2, ..., t_n)$, siendo $t_i \in \{1, ..., n\}$, con $t_i \neq t_j$, $\forall\ i \neq j$

  - t_i es el número de tarea asignada a la persona i.

  - Da lugar a un árbol permutacional.

- Existe la alternativa simétrica con el vector de asignaciones desde el punto de vista de las tareas.

4 – Ejemplos: Planificación de tareas.

Calculo de cotas: opción 1, 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:

$$BE(x) = (CI(x)+CS(x))/2.$$

4 – Ejemplos: Planificación de tareas.

Calculo de cotas: opción 2, estimaciones con 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 (Greedy).

- Cota superior: Asignar las tareas con mayor beneficio (aunque se repitan).

- Estimación del beneficio: La media de las cotas:

$$BE(x) = (CI(x)+CS(x))/2.$$

4 – Ejemplos: Planificación de tareas.

Estrategia de poda: Problema de maximización

✓ Variable de poda C: Valor de la mayor cota inferior o solución final del problema encontrada hasta ahora.

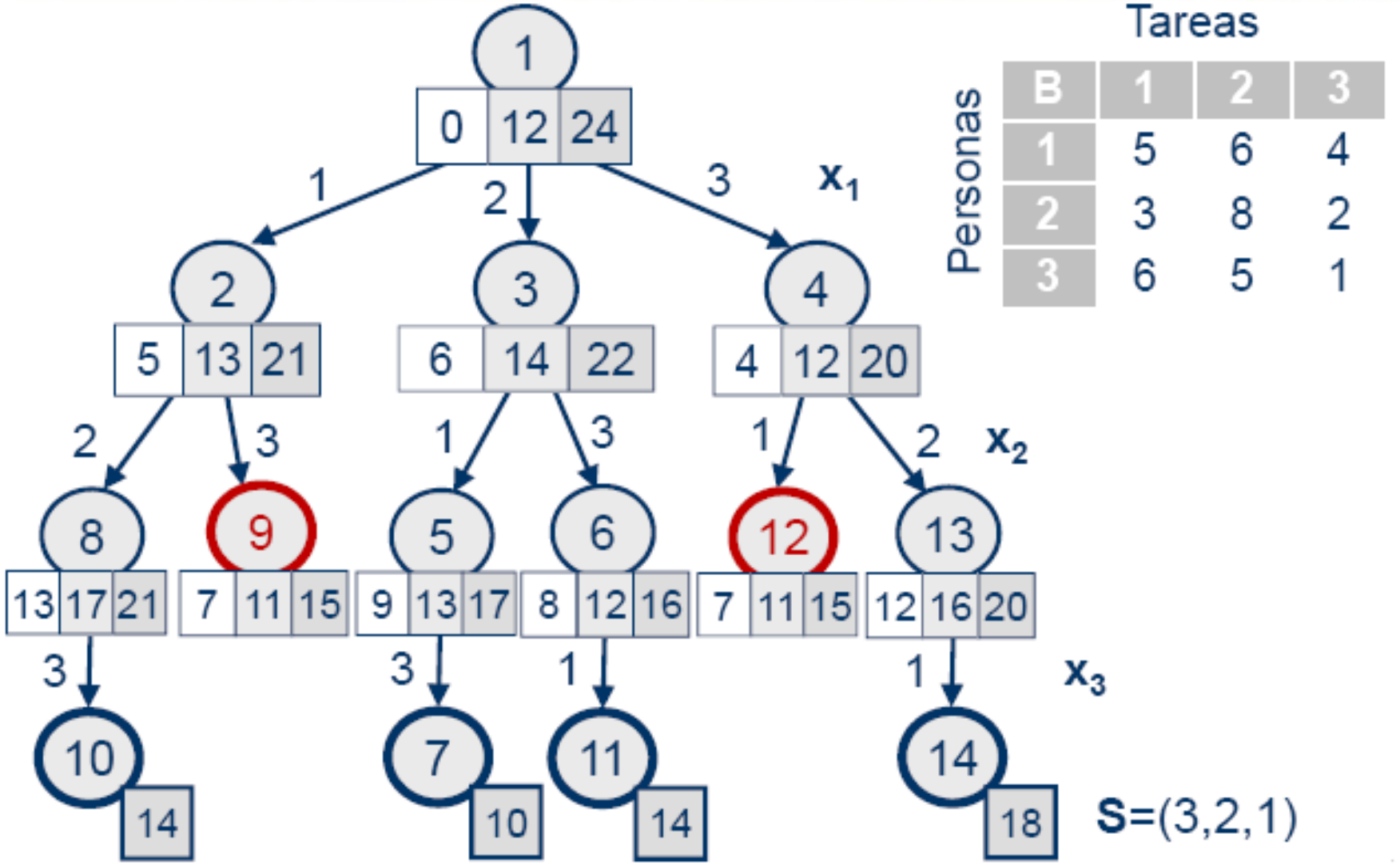
✓ Condición de poda: Podar el nodo i si $CS(i) \leq C$.

Estrategia de ramificación: MB-LIFO

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

4 – Ejemplos: Planificación de tareas.

Ejemplo: opción 1 – cotas triviales -

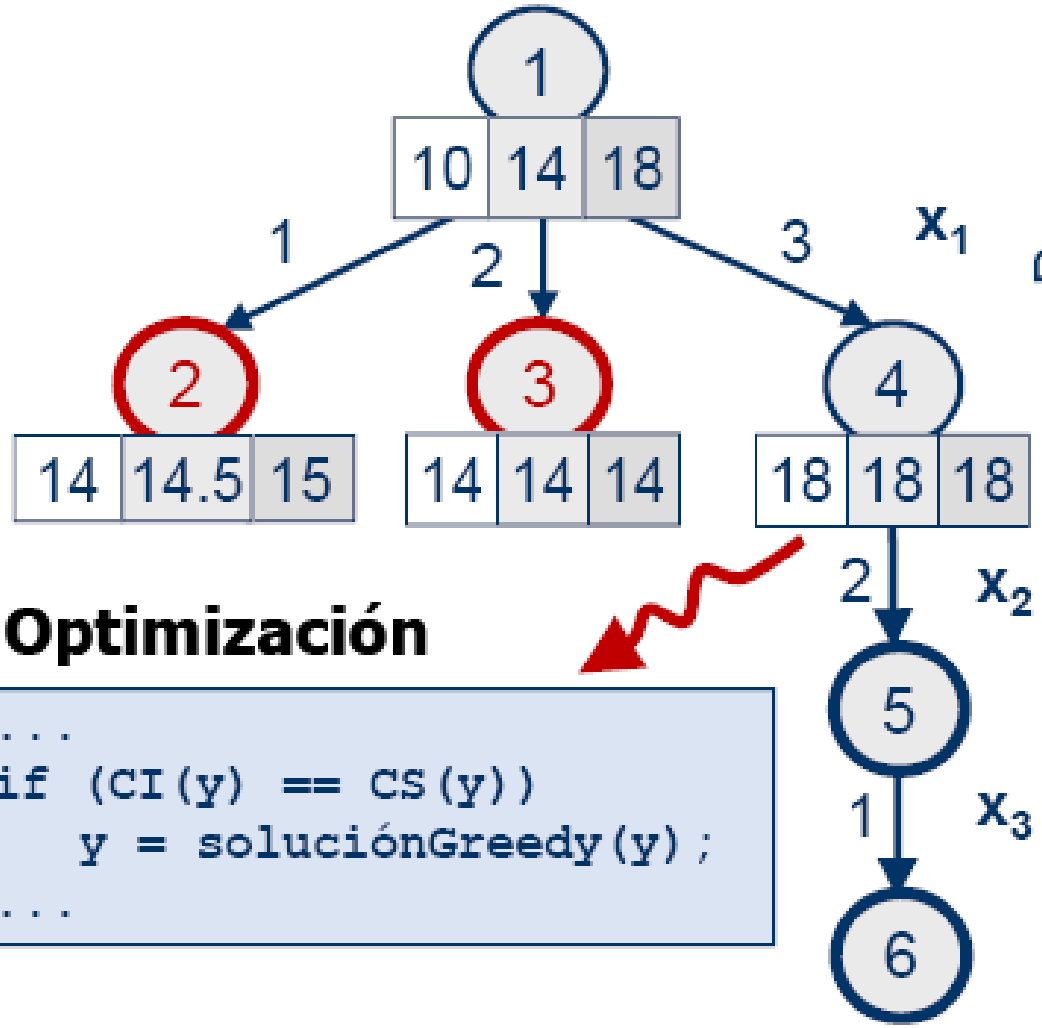


Tareas

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

4 – Ejemplos: Planificación de tareas.

Ejemplo: opción 2 – cotas con algoritmo Greedy -



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);
...
```

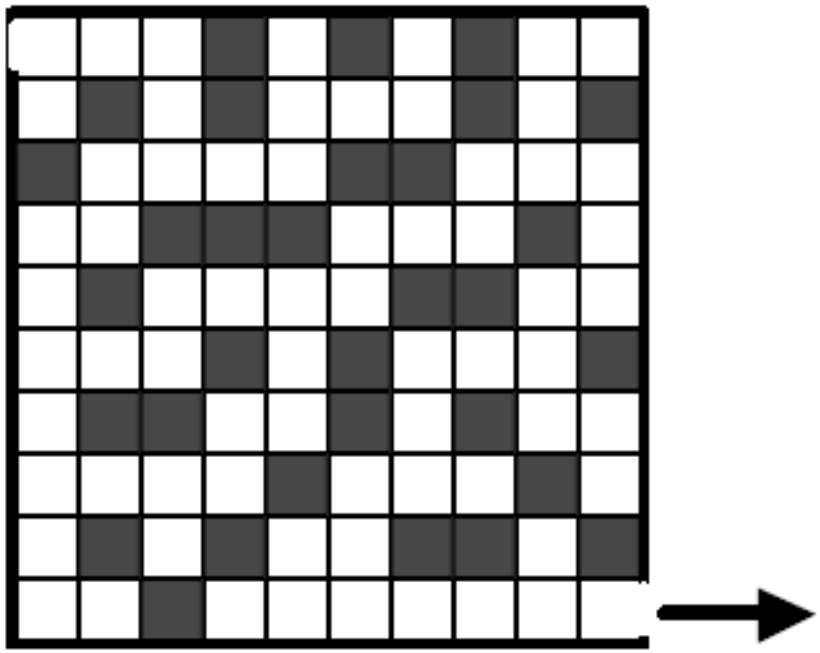
C	LNV
10	1
18	2 3
18	3

4 – Ejemplos: Otros ejemplos.

El problema de la salida de un laberinto:

Laberinto: Matriz de n·n, de casillas *libres* u *ocupadas* por pared.

- Es posible pasar de una casilla a otra sólo en vertical u horizontal.
- Se debe ir de la casilla (1, 1) a la (n, n). – Este es un caso especial –



4 – Ejemplos: Otros ejemplos.

El problema de la salida de un laberinto:

- El problema era abordable utilizando backtracking, por lo que también lo será con branch and bound.

- Es necesario definir la función coste de un nodo: se calculará como la suma del número de pasos dados hasta él, más la distancia Manhattan desde él hasta la salida (es el valor del camino teórico más corto que pasaría por ese nodo, sabiendo que quizá no exista tal camino). En nuestro caso, la función más importante es la de coste (ya que nos va a permitir aplicar seleccionar el nodo a ramificar utilizando MB-LIFO.

- La cota inferior será igual al coste estimado (ya que representa el costo de llegar al nodo, más el coste de llegar desde él hasta la salida sin obtáculos.

- La cota superior puede ser ∞ o nxn. No tiene función en nuestro caso.

4 – Ejemplos: Otros ejemplos.

El problema de la salida de un laberinto: Ejemplo

Sea el siguiente laberinto, donde:

- La casilla de partida es la (1,1). - La casilla de llegada es la (7,7).
- Las que tienen -1 representan muros.
- Las casillas con 0 representan un posible elementos del camino no explorado.

1	-1	0	0	0	0	-1
2	-1	0	-1	-1	0	-1
3	-1	0	-1	-1	0	0
4	5	6	0	-1	0	-1
-1	6	-1	0	-1	0	-1
-1	7	-1	0	-1	0	-1
0	8	-1	0	-1	0	0

- Las casillas con número mayor de 0, representan casillas exploradas (el valor representa el coste para llegar a él desde el inicio).
- La casilla con recuadro, representa la casilla en la que encontramos en el nodo actual de partida es la (1,1).
- La zona sombreada representa el camino Manhattan del nodo en expansión a la salida.

4 – Ejemplos: Otros ejemplos.

El problema de la salida de un laberinto: Ejemplo

En la posición marcada (nodo 4-3):

Coste = coste llegar + Camino Manhattan = 6 + 7 = 13

donde

- Las casillas numeradas del 1 al 5 representan el camino recorrido hasta llegar a la que está en estudio, la posición recuadrada.
- Se muestra sombreado el camino Manhattan a la salida, incluyendo cuatro -1 (muro) y tres accesibles (0).
- Si se utiliza el esquema MB-LIFO y la generación de nodos es (arriba, derecha, abajo, izquierda), como en las agujas del reloj; también hay nodos vivos por debajo del nodo en estudio (se supone que se ha pasado por ellos), pero al llegar a la esquina (7,1), su coste pasa a ser 14 y se adelanta en la exploración el nodo en estudio.

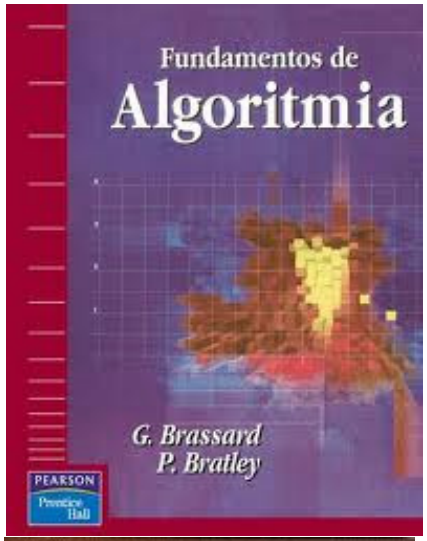
4 – Ejemplos: Otros ejemplos.

El problema de la salida de un laberinto: forma de trabajo

- Se va explorando el árbol, utilizando la función de coste para seleccionar el nodo en expansión según el mecanismo MB-LIFO.
- Al costo global C, se le da el valor infinito o una cota superior segura (n*n).
- Cuando se llega por primera vez a la salida, se pueden comenzar a podar los nodos que poseen una cota inferior (igual al coste estimado) superior o igual a la cota superior ya encontrado.
- Cada nodo debe contener la información suficiente para saber cómo se llegó a él y cómo se puede continuar avanzando (copia del laberinto, con los pasos dados hasta llegar a él).

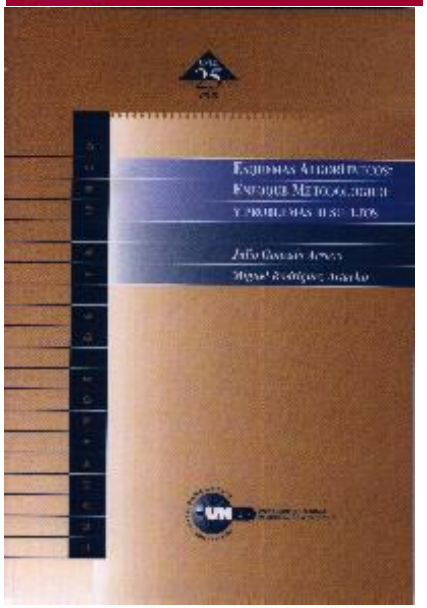
El algoritmo va apuntando la mejor solución encontrada hasta el momento (mejorSol), la cual se inicializa con un coste infinito o n*n.

Material a estudiar



Fundamentos de algoritmia
G. Brassard, T. Bratley
Prentice Hall, D.L. 2004

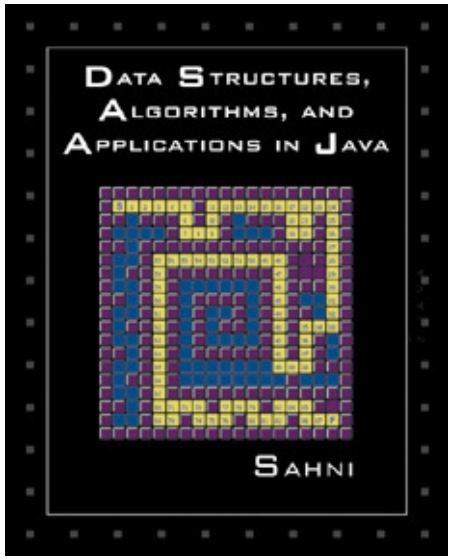
Biblioteca: 519 BRA fun
Capítulo 9 (apartados 9-7)

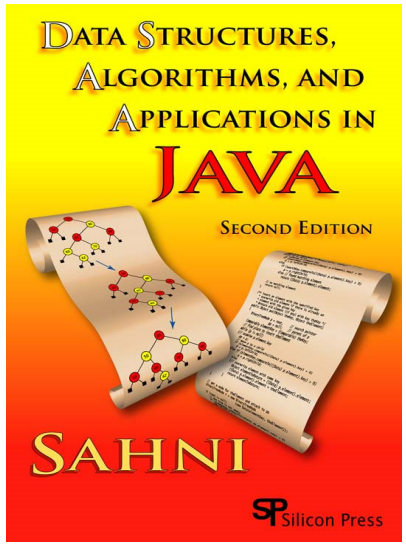


Esquemas algorítmicos. Enfoque metodológico y problemas resueltos
Julio Gonzalo Arroyo y Miguel Rodríguez Artacho)
ISBN: 84-362-3622-X

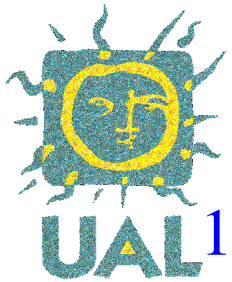
Biblioteca: 519 GON esq
Capítulo 4 (sobre todo para ejemplos.)

Material a estudiar





Data Structures, Algorithms, and Applications in Java
Sartaj Sahni
McGraw Hill, 2000 / Silicom Press, 2005
Biblioteca: Agotado – Esperando 2ª edición. Capítulo 22.



Ejercicios recomendados - Libres

1. Especificar el esquema algorítmico Branch & Bound del problema del viajante.

2. Especificar el esquema algorítmico Branch & Bound del problema de planificación de tareas.

3. Especificar el esquema algorítmico Branch & Bound del problema del laberinto.