Diseño y Análisis de Algoritmos

TEMA 6. RAMIFICACIÓN Y PODA

Universidad
Rey Juan Carlos

Tema 6. Ramificación y Poda (*B&B*)

Introducción

- Introducción
- Estrategia general
- Esquema de la técnica
- Ejemplos
  - Asignación de tareas
  - Mochila 0/1

Introducción

- Técnica de exploración en grafos dirigidos implícitos (usualmente **árboles**)
- Se suele utilizar para encontrar la solución a un **problema de optimización** con restricciones.
- Es aplicable a problemas en los que la solución puede expresarse en forma de ***n*-tupla** de componentes
- Posee **similitudes** con la técnica de Backtracking,
- *Branch & Bound* intenta hacer una búsqueda más "**inteligente**"

Introducción

- Para ello, emplea **cotas** para podar las ramas del árbol que no llevan a la solución óptima:
  - Calcula en cada nodo una cota del posible **mejor valor** alcanzable desde ese nodo
  - Si este valor **es peor** que el de la **mejor solución alcanzada** hasta el momento $\Rightarrow$ se poda la rama
- El hecho de podar algunas ramas, evita **explorar todas** las soluciones factibles.
- Sólo se puede realizar la **poda** cuando ya se tiene **una solución** o se dispone de una **cota teórica**

Tema 6. Ramificación y Poda (*B&B*)

Estrategia general

Terminología B&B:

- **Nodo vivo**. Aquél que no ha sido podado y que puede ser ramificado
- **Nodo muerto**. Aquél del que no se van a generar más hijos porque:
  - Se llega a una solución
  - No genera nuevas soluciones factibles
  - No genera mejores soluciones que la mejor conocida hasta el momento
- **Nodo en curso (o en expansión)**. Aquél que se selecciona para ramificarlo

Tema 6. Ramificación y Poda (*B&B*)

Estrategia general

Estrategias de exploración B&B:

- La implementación B&B suele ser **iterativa**
- Se utiliza una **estructura de datos** en la que se almacenan los **nodos vivos**
- La estructura auxiliar depende de la **estrategia de ramificación** empleada:
  - LIFO: pila (exploración en profundidad)
  - FIFO: cola (exploración en anchura)
  - PQ: cola de prioridad o montículo (exploración por el más prometedor)

Tema 6. Ramificación y Poda (*B&B*)

Estrategia general

Diferencias entre *Backtracking* y *Branch & Bound:*

- Con respecto al **nodo en curso**:
  - BT: Según se genera un hijo, éste pasa a ser el nodo en curso
  - B&B: Se generan todos los hijos del nodo en curso antes de decidir cuál va a ser el siguiente nodo en curso (estrategias LIFO, FIFO, PQ).
- Con respecto a los **nodos vivos**:
  - BT: Los únicos nodos vivos son los que están en el camino de la raíz al nodo en curso
  - B&B Puede haber más nodos vivos.

Tema 6. Ramificación y Poda (*B&B*)

Esquema de la técnica

- **Selección**: elige el nodo vivo que será ramificado (dependerá de la estrategia)
- **Ramificación**: se generan los hijos del nodo seleccionado (sólo tuplas prometedoras)
- **Cálculo de cotas**: Para cada nodo, se calcula una cota del posible mejor valor alcanzable desde ese nodo
- **Poda**: se podan los nodos generados en la etapa anterior que no van a conducir a una solución mejor que la mejor conocida hasta ahora

Esquema de la técnica

Gestión de nodos vivos:

- Los nodos que **no son podados** pasan a formar parte del conjunto de **nodos vivos**
- El nodo en **curso** se selecciona en función de la **estrategia elegida**
- El algoritmo **finaliza** cuando:
  - Se agota el conjunto de nodos vivos (solución **óptima**)
  - Se encuentra una solución que satisface un **umbral de calidad**

Esquema de la técnica

- B&B es **efectivo** si posee una función de coste adecuada, es decir:
  - Podar lo máximo posible
  - Cálculo eficiente (baja complejidad computacional)
- Para empezar a podar es necesario tener una **solución inicial** o una **cota general** del problema
- Se suele utilizar **un algoritmo voraz** para tener una primera solución con la que empezar a comparar

Tema 6. Ramificación y Poda (*B&B*)

Esquema de la técnica

```
función RyP()
sol ← emptySol() {Tupla parcial inicialmente vacía}
finalSol ← upperBound() {algoritmo voraz o equivalente}
upBound ← cost(finalSol)
q ← emptyQueue()  {Pila, cola o cola de prioridad}
enqueue(sol,q)
mientras not emptyQueue(q) hacer
    sol ← first(q)
   dequeue(q)
    si isSol(sol) entonces
       si cost(sol) < upBound entonces {Para problemas de minimización}
          finalSol ← sol
           upBound ← Coste(sol)
       fsi
    sino
       si lowerBound(sol) < upBound entonces
         para cada child en complete(sol) hacer
            si isFeasible(child) y (lowerBound(child) < upBound) entonces
               enqueue(hijo,q)
            fsi
         fpara
       fsi
    fsi
fmientras
devolver finalSol
ffunción
```

Tema 6. Ramificación y Poda (*B&B*)

Esquema de la técnica

- `sol`: tupla (nodo del árbol de exploración)
- `emptySol()`: genera una tupla vacía
- `upperBound()`: construye una solución de calidad utilizando un algoritmo voraz
- `isSol()`: indica si una tupla es una solución
- `lowerBound(sol)`: estima una cota de la calidad de la mejor solución posible que se podría encontrar ramificando ese nodo (no tiene por qué ser factible)
- `complete()`: calcula el conjunto de posibles sucesores dada una tupla (componentes disponibles)
- `isFeasible()`: determina si una tupla es factible (o puede conducir a una solución factible)

Dadas n tareas y n personas, asignar a cada persona una tarea minimizando el coste de la asignación total. Se dispone una matriz de tarifas que determina el coste de asignar a cada persona una tarea

	1	**2**	**3**
a	4	7	3
b	2	6	1
c	3	9	4

- Si el agente $1 \leq i \leq n$, se le asigna la tarea $1 \leq j \leq n$, el coste será c_{ij}

- Matriz de costes para el problema de la asignación:

	1	**2**	**3**	**4**
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

- Obtener una cota superior del coste (primera solución)
  - P. ej., los elementos de la diagonal ppal.:

$$a \leftarrow 1,\ b \leftarrow 2,\ c \leftarrow 3,\ d \leftarrow 4$$

$$11 + 15 + 19 + 28 = 73$$

$$\text{solución óptima} \leq \text{cota calculada}$$

Tema 6. Ramificación y Poda (*B&B*)

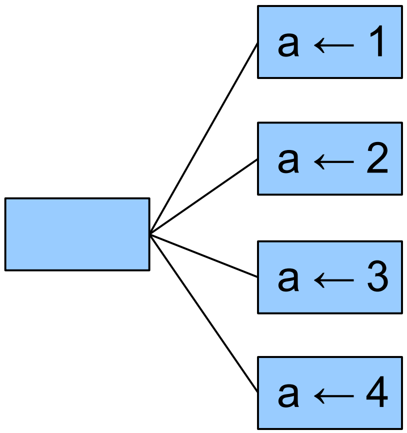
Ejemplo. Asignación de tareas

- Árbol de expansión: En cada paso se asigna una tarea a un agente.
  - **Raíz**: no se ha realizado ninguna asignación
  - **Nivel *k***: se han asignado *k* tareas a *k* agentes
- Cálculo de cotas:
  - Se calcula para **cada nodo**
  - La cota da idea del coste de la mejor solución que se **podría** obtener **a partir de ese nodo**
  - Pueden determinar la **eliminación** de caminos (poda)
  - Nuestro problema: para calcular una cota en el nivel *k* se consideran las **tareas con menor coste** de las disponibles

Ejemplo. Asignación de tareas

Empezamos asignando al agente ***a*** a las diferentes tareas

Cota sup: 73



11 + cota1 = **11** + 13 + 17 + 14 = 55

12 + cota2 = **12** + 13 + 11 + 17 = 53

18 + cota3 = **18** + 14 + 11 + 14 = 58

40 + cota4 = **40** + 13 + 11 + 14 = 78*

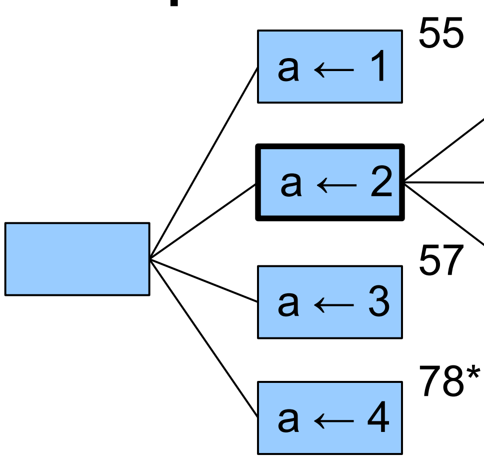
* 78 > cota superior ⇒ no se explorará, no se encuentra el óptimo en ella

	1	**2**	**3**	**4**
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Ejemplo. Asignación de tareas

A continuación se explora el nodo más prometedor: cota 53

Cota sup: 73



12 + **14** + cota21 = **12** + **14** + 19 + 20 = 65

12 + **13** + cota23 = **12** + **13** + 11 + 17 = 53

12 + **22** + cota24 = **12** + **22** + 11 + 17 = 62

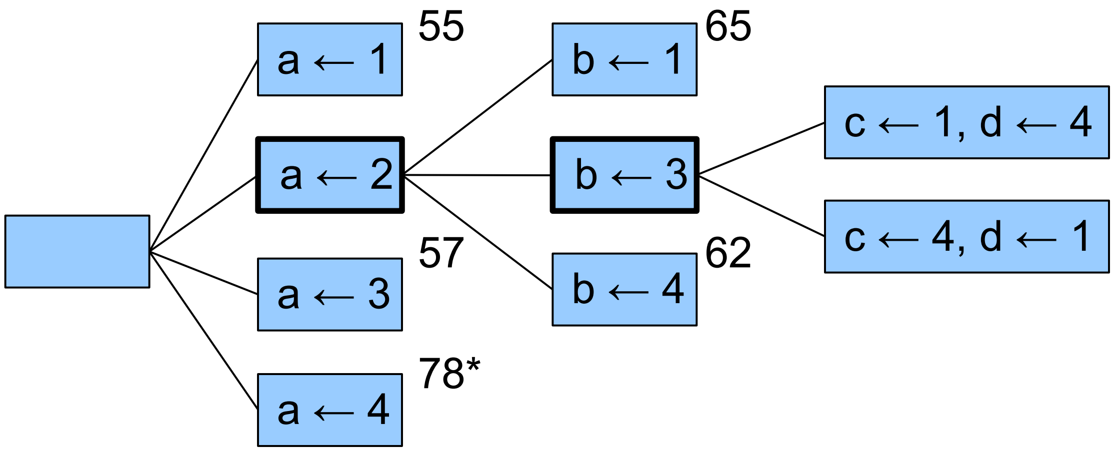
El nodo más prometedor es el que tiene una cota 53

	1	**2**	**3**	**4**
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Ejemplo. Asignación de tareas

A continuación se explora el nodo más prometedor: cota 53

Cota sup: 73



(**12**+**13**+**11**+**28**) = **64**

(**12**+**13**+**23**+**17**) = 65

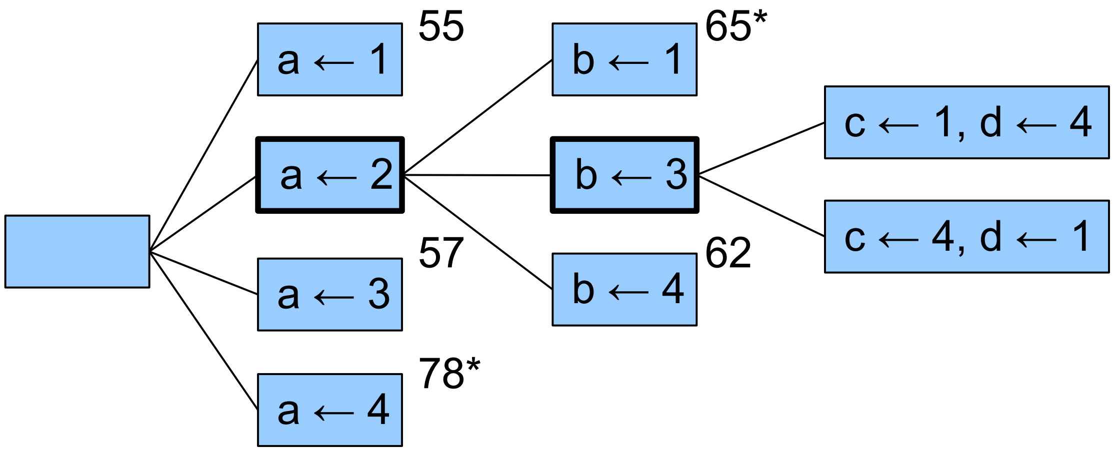
La solución encontrada será nuestra nueva cota superior: 64

Como consecuencia, descartamos el nodo (a ← 2, b ← 1), que excede la nueva cota superior

	1	**2**	**3**	**4**
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Situación después de encontrar la solución anterior:

Cota sup: 64

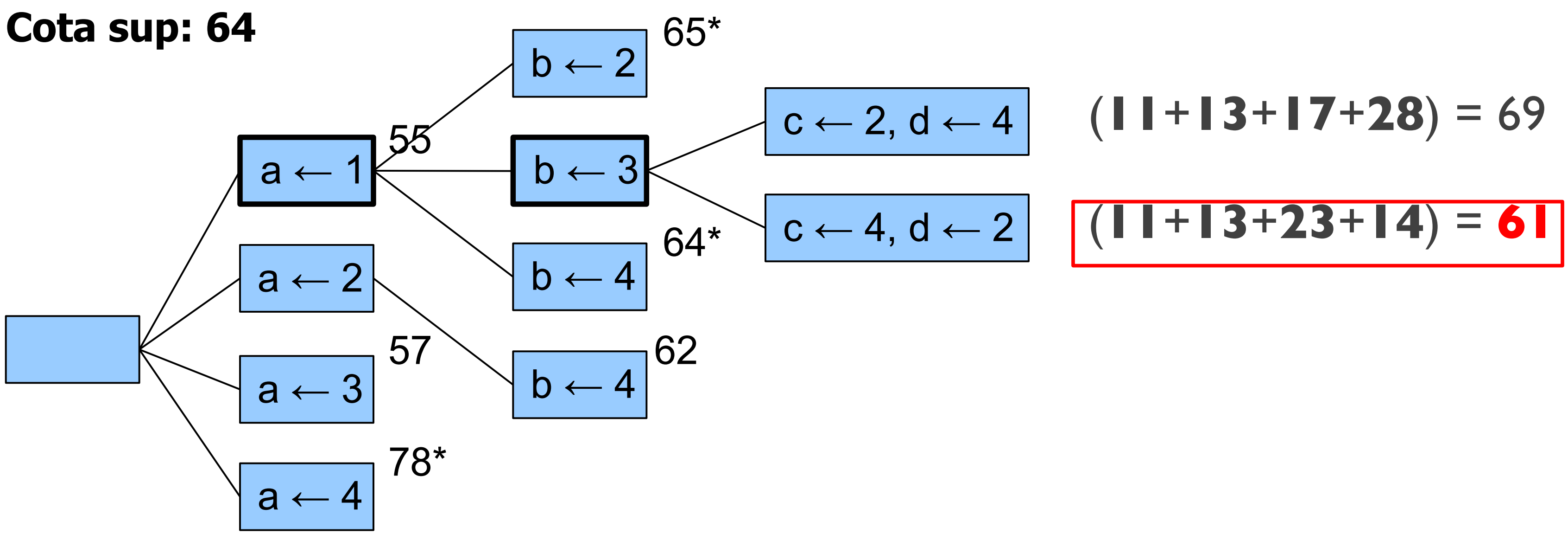


(**11**+**12**+**13**+**28**) = **64**

(**17**+**12**+**13**+**23**) = 65

Ejemplo. Asignación de tareas

a→1 tiene la cota más prometedora y se sigue la exploración a partir de él



La solución encontrada será la nueva cota superior: 61

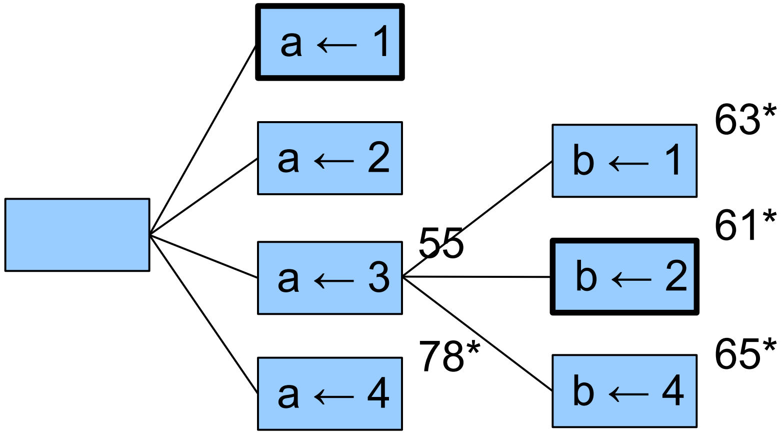
Como consecuencia, descartamos el nodo (a ← 2, b ← 4)

	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Ejemplo. Asignación de tareas

a→3 tiene la cota más prometedora y se sigue la exploración a partir de él

Cota sup: 61



18 + **14** + cota31 = **18** + **14** + 17 + 14 = 63

18 + **15** + cota32 = **18** + **15** + 11 + 17 = 61

18 + **22** + cota34 = **18** + **22** + 11 + 14 = 65

Hemos encontrado la solución óptima, reduciendo el coste que supondría una exploración completa

	1	**2**	**3**	**4**
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Ejemplo. Asignación de tareas

- Se **parte** de una **solución inicial**
- La **exploración** del árbol de búsqueda se realiza examinando el nodo **más prometedor** en cada momento
- Las **cotas** de los nodos se calculan sumando el mínimo coste de las tareas no asignadas al coste parcial de la solución (suma del coste de las tareas no asignadas)
- Es necesaria una **estructura auxiliar** para almacenar ordenadamente los posibles nodos a explorar

- El **coste** depende de la **calidad** de la primera cota seleccionada
- Si la cota es buena:
  - se examinan **menos nodos**
  - la solución se encontrará en **menos pasos**
- En el **caso peor**, una cota buena puede que no **nos permita** podar muchas ramas del árbol
- Los **cálculos** asociados a la obtención de las cotas tienen asimismo un **coste**
- En general, suele ser rentable obtener una **buena cota**

Tema 6. Ramificación y Poda (*B&B*)

Ejemplo. Asignación de tareas

```
función TareasYAgentes(matrizCostes: matriz):TSolucion
q ← emptyQueue {Pila, cola o cola de prioridad}
finalSol ← initSol()
upperBound ← cost(finalSol)
v ← VectorMinimum(matrizCostes);
enqueue(sol,q)
mientras no emptyQueue(q) hacer
          sol ← first(q)
          dequeue(q)
          si isSol(sol) entonces
             si cost(sol) < upperBound entonces
                finalSol ← sol
                upperBound ← cost(sol)
             fsi
          sino
             si lowerBound(sol) < upperBound entonces
               para cada child en complete(sol) hacer
                  si isFeasible(hijo) y lowerBoud(hijo,v) < upperBound entonces
                     InsertarPorPrioridad(hijo,q)
                  fsi
               fpara
             fsi
          fsi
fmientras
devolver solucionFinal
ffunción
```

Tema 6. Ramificación y Poda (*B&B*)

Ejemplo. Asignación de tareas

- `lowerBound()`: devuelve la suma de los costes de las tareas asignadas y los costes mínimos de las no asignadas
- `upperBound()`: almacena el coste de la mejor solución encontrada
- `complete()`: toma la primera tarea sin asignar y genera tantos nodos como agentes hay disponibles para esa tarea
- `isFeasible()`: la solución no puede tener ni tareas ni agentes repetidos

Cada nodo del árbol debe contener:

- Vector con las asignaciones realizadas (tupla parcial)
- Una lista de tareas aún no asignadas (elementos disponibles)
- Último agente asignado
- Coste de esa solución (parcial)

```
función isSol(sol: tupla): boolean;
   devolver (ultimoAgenteAsignado(sol) = n)
ffunción;
```

```
función lowerBound(sol:tupla;w:vectorMinimos):
entero;
   devolver
```

$$Coste(sol) + \sum_{i=ultimaAsignacion(sol)+1}^{n} \min(w[i,j])$$

```
ffunción;
```

```
función complete(e:tupla): ListaTuplas;
{Genera todas las alternativas posibles}
ffunción;
```

Tema 6. Ramificación y Poda (*B&B*)

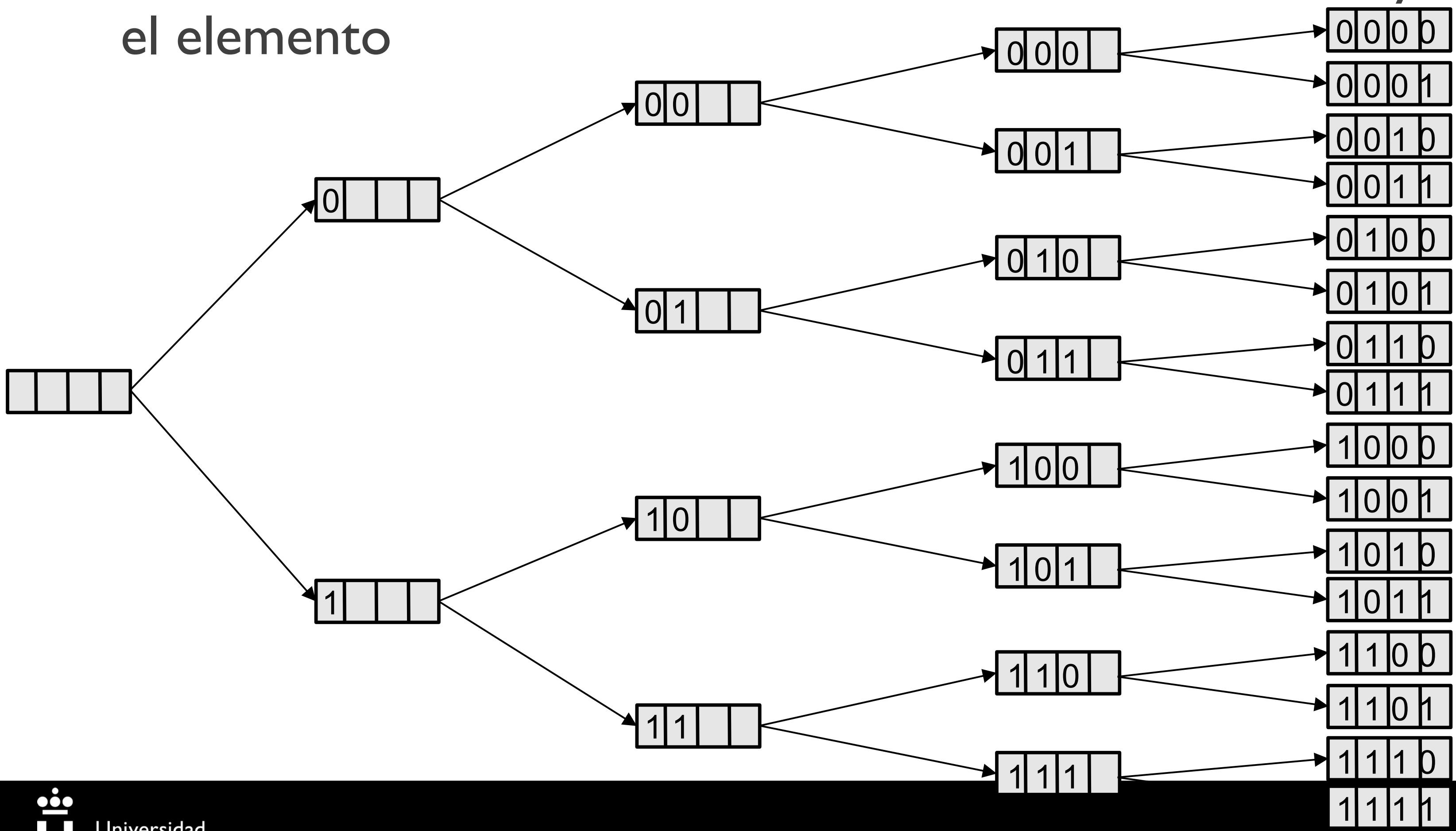
Ejemplo. Mochila 0-1

Tenemos *n* objetos y una mochila. Cada objeto i tiene un peso $w_i > 0$ y un valor $v_i > 0$. La mochila puede llevar un peso que no sobrepase *W*. Se desea llenar la mochila maximizando el valor de los objetos transportados. Los objetos no pueden partirse en fracciones.

$$\max \sum_{i=1}^{n} x_i v_i \quad \text{con la restricción} \quad \sum_{i=1}^{n} x_i w_i \leq W$$

Ejemplo. Mochila 0-1

- Árbol de exploración binario:
  - En cada nivel del árbol se toma la decisión de si se incluye o no el elemento



Tema 6. Ramificación y Poda (*B&B*)

Ejemplo. Mochila 0-1

Estrategias de resolución de problemas aplicables al problema de la mochila 0-1:

- **Voraz**: no asegura que se encuentre la mejor solución sin reconsiderar decisiones previas
- **Backtracking**: para encontrar la mejor solución hay que explorar todo el subespacio de factibilidad
- **Branch & Bound**: Permite incrementar la eficiencia de la búsqueda mediante el uso de funciones de poda

Tema 6. Ramificación y Poda (*B&B*)

Ejemplo. Mochila 0-1

Ejemplo:

$n = 4, \quad W = 15$

v	10	10	12	18
w	2	4	6	9
v/w	5.0	2.5	2.0	2.0

- Los objetos se ordenan en **orden decreciente** del ratio beneficio/peso (facilita el cálculo de la función de coste)
- El esquema de B&B (pseudocódigo) que hemos visto, está diseñado para problemas de **minimización** y este problema es de **maximización**
- La adaptación del esquema es directa. Dos opciones:
  - **Adaptar** el algoritmo: Cambiar desigualdades
  - **Redefinir** el problema $\min\left(-\sum_{i=1}^{n} x_i v_i\right)$

Solución inicial:

- Empleamos un algoritmo voraz (no es la única opción)
- Se añade un elemento a la solución siempre que, en total, no se supere el valor umbral
- Como resultado, obtenemos la solución inicial:
  - x_{inic} =(1,1,1,0)
  - Coste(x_{inic}) = 32 Cota Solución Inicial
  - Peso(x_{inic}) = 12

Ejemplo. Mochila 0-1

Sea B_k el beneficio obtenido por la tupla *k*:

$$B_k = \sum_{i=1}^{k} x_i v_i$$

Llamemos B_M a la cota de beneficio máximo al que podemos aspirar desde el nodo *k*. La estimamos del siguiente modo:

$$B_M = B_k + \left(W - \sum_{i=1}^{k} x_i w_i \right) \frac{v_{k+1}}{w_{k+1}}$$

Tema 6. Ramificación y Poda (*B&B*)

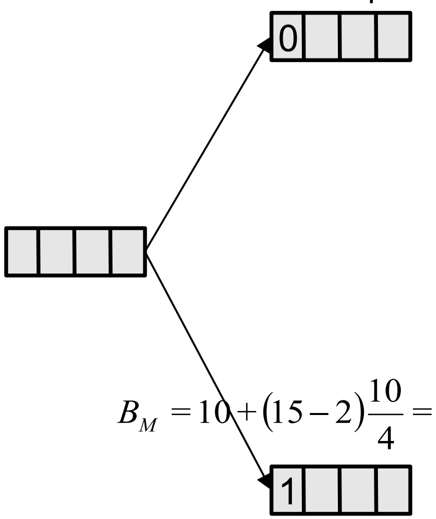
Ejemplo. Mochila 0-1

	$n = 4,\ W = 15$			
v	10	10	12	18
w	2	4	6	9
v/w	5.0	2.5	2.0	2.0

Cota solución: 32

$$B_M = B_k + \left(W - \sum_{i=1}^{k} x_i w_i \right) \frac{v_{k+1}}{w_{k+1}}$$

$$B_M = 0 + (15 - 0)\frac{10}{4} = 37.5$$



$$B_M = 10 + (15 - 2)\frac{10}{4} = 42.5$$

Problema de maximización $\Rightarrow$ el [1, , ,] es el nodo más prometedor

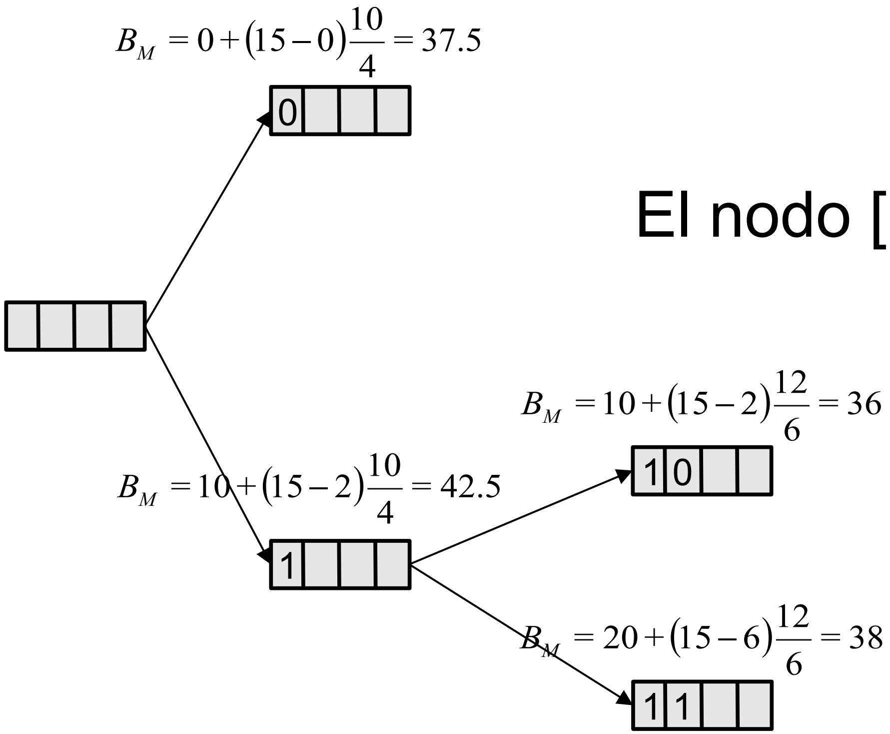
Tema 6. Ramificación y Poda (*B&B*)

Ejemplo. Mochila 0-1

	$n = 4,\ W = 15$			
v	10	10	12	18
w	2	4	6	9
v/w	5.0	2.5	2.0	2.0

Cota solución: 32

$$B_M = B_k + \left(W - \sum_{i=1}^{k} x_i w_i \right) \frac{v_{k+1}}{w_{k+1}}$$



El nodo [1,1 , ,] es el más prometedor

	$n = 4$, $W = 15$			
v	10	10	12	18
w	2	4	6	9
v/w	5.0	2.5	2.0	2.0

Cota solución: 32

$$B_M = B_k + \left(W - \sum_{i=1}^{k} x_i w_i\right)\frac{v_{k+1}}{w_{k+1}}$$

Los dos nodos son igualmente prometedores. Supongamos que el [1,1,0,] ha entrado antes en la cola

$B_M = 0 + (15-0)\frac{10}{4} = 37.5$ [0| | |]

[| | |]

$B_M = 10 + (15-2)\frac{10}{4} = 42.5$ [1| | |]

$B_M = 10 + (15-2)\frac{12}{6} = 36$ [1|0| |]

$B_M = 20 + (15-6)\frac{12}{6} = 38$ [1|1| |]

$B_M = 20 + (15-6)\frac{18}{9} = 38$ [1|1|0|]

$B_M = 32 + (15-12)\frac{18}{9} = 38$ [1|1|1|]

Tema 6. Ramificación y Poda (*B&B*)

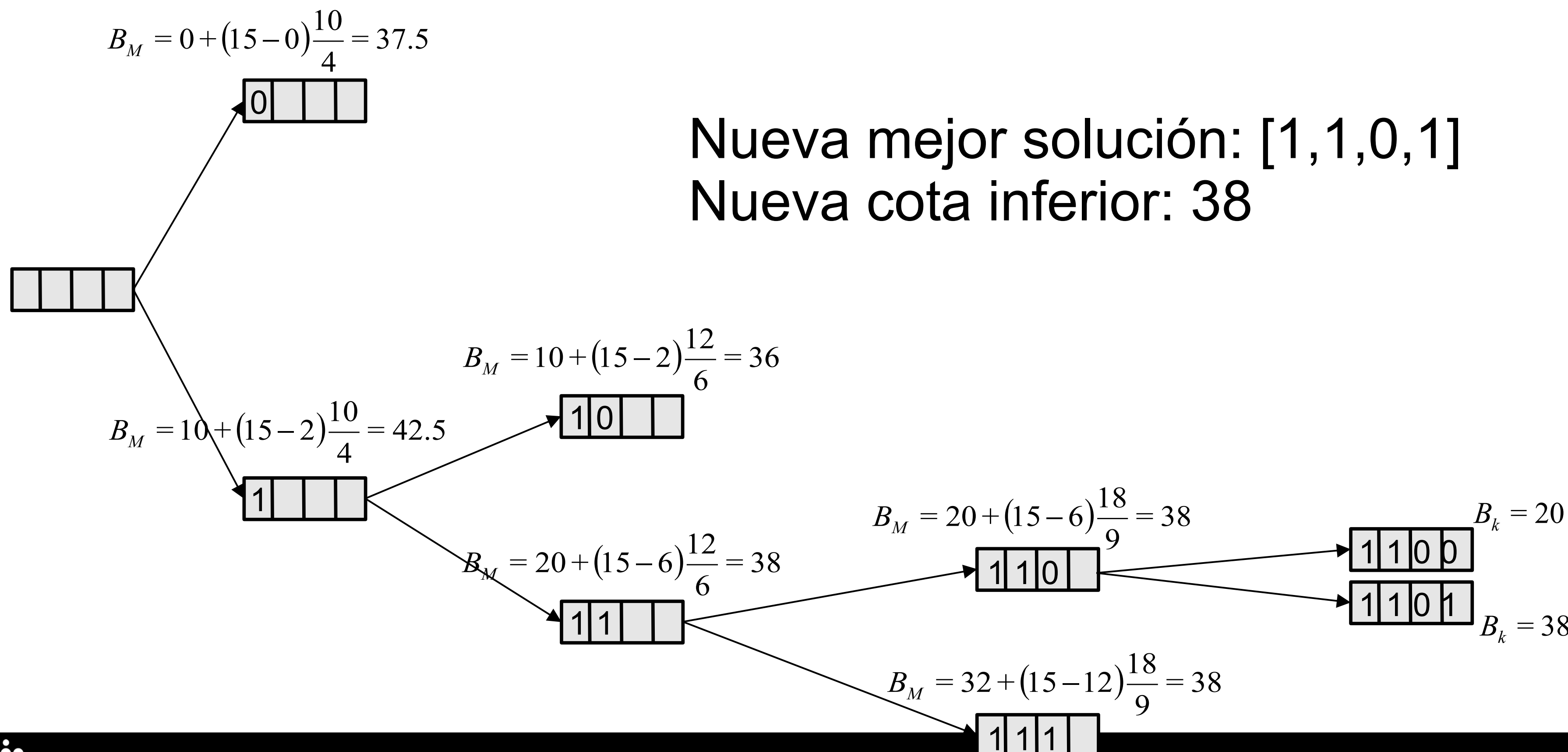
Ejemplo. Mochila 0-1

	$n = 4,\ W = 15$			
v	10	10	12	18
w	2	4	6	9
v/w	5.0	2.5	2.0	2.0

Cota solución: 32

$$B_M = B_k + \left(W - \sum_{i=1}^{k} x_i w_i\right)\frac{v_{k+1}}{w_{k+1}}$$

Nueva mejor solución: [1,1,0,1]
Nueva cota inferior: 38



Tema 6. Ramificación y Poda (*B&B*)

Ejemplo. Mochila 0-1

	$n = 4,\ W = 15$			
v	10	10	12	18
w	2	4	6	9
v/w	5.0	2.5	2.0	2.0

Cota solución: 38

$$B_M = B_k + \left(W - \sum_{i=1}^{k} x_i w_i \right) \frac{v_{k+1}}{w_{k+1}}$$

Situación después de actualiza la "cota solución": se matan todos los nodos vivos y se termina la búsqueda

$$B_M = 0 + (15 - 0)\frac{10}{4} = 37.5$$

[0| | |] *

$$B_M = 10 + (15 - 2)\frac{10}{4} = 42.5$$

[1| | |]

$$B_M = 10 + (15 - 2)\frac{12}{6} = 36$$

[1|0| |] *

$$B_M = 20 + (15 - 6)\frac{12}{6} = 38$$

[1|1| |]

$$B_M = 20 + (15 - 6)\frac{18}{9} = 38$$

[1|1|0|]

$$B_M = 32 + (15 - 12)\frac{18}{9} = 38$$

[1|1|1|] *

$B_k = 20$ [1|1|0|0]

$B_k = 38$ [1|1|0|1]

Tema 6. Ramificación y Poda (*B&B*)

Ejemplo. Mochila 0-1

```
función RyP(sol) {Modificado para un problema de maximización}
   q ← emptyQueue {Pila, cola o cola de prioridad}
   finalSol ← initSol()
   lowerBound ← cost(finalSol)
   enqueue(sol,q)
   mientras no emptyQueue(q) hacer
      sol ← Primero(q)
      dequeue(q)
      si isSol (sol) entonces
         si cost(sol) > lowerBound entonces
            finalSol ← sol
            lowerBound ← cost(sol)
         fsi
         sino
            si upperBound(sol) > lowerBound entonces
              para cada child en complete(sol) hacer
                 si isfeasible(child) y upperBound(child) > lowerBound entonces
                    priorityEnqueue(child,q)
                 fsi
              fpara
            fsi
         fsi
fmientras
devolver finalSol
ffunción
```

- `upperBound()` ⇒ calcula la suma del beneficio de los elementos que ya están en la mochila más el beneficio resultante de llenar el resto de la mochila con el siguiente elemento (suponemos que se puede fraccionar, pero sólo en el cálculo de cotas)
- `lowerBound()` ⇒ el coste de la mejor solución encontrada
- `complete()` ⇒ genera dos hijos: uno con el siguiente elemento y otro sin él.
- `isFeasible()` ⇒ Sólo son válidas aquellas tuplas cuyo coste sea menor o igual que W.
- `isSol()` ⇒ devuelve verdadero cuando todos los objetos han sido considerados y falso en caso contrario

Tema 6. Ramificación y Poda (*B&B*)

Ejemplo. Mochila 0-1

Cada nodo del árbol debe contener:

- Vector con los elementos que se han incluido en la mochila
- Peso acumulado
- Beneficio acumulado
- Entero que indica cuál es el siguiente objeto a considerar