Ramificación y poda (*branch & bound*)

Simona Bernardi

Universidad de Zaragoza

Presentación adaptada de la original de J. Campos

Resumen

- Método general
- Aplicación
  - El juego del 15
  - Problemas de optimización
    - Un problema de planificación de tareas a plazo fijo
    - El problema de la mochila 0-1
    - El problema del viajante de comercio

Ramificación

- Al igual que los métodos de búsqueda con retroceso:
  - Se aplica a problemas de optimización con restricciones
  - Se genera el espacio de soluciones, organizandólo en un árbol (en general en un grafo)
  - No se genera el espacio de soluciones completo, sino que se podan bastantes estados

Terminología

- Nodo **vivo**: nodo del espacio de soluciones del que no se han generado aún todos sus hijos
- Nodo **muerto**: nodo del que no se van a generar más hijos porque
  - no hay más
  - no es completable, es decir viola las restricciones
  - no producirá una solución mejor que la solución actual
- Nodo **en expansión**: nodo del que se están generando hijos

Diferencia con el método de búsqueda con retroceso

- Búsqueda con retroceso: tan pronto como se genera un nuevo hijo del nodo en expansión, este hijo pasa a ser el nodo en curso
- Ramificación y poda: se generan todos los hijos del nodo en expansión antes de que **cualquier otro nodo vivo** pase a ser el nuevo nodo en expansión

Consecuencia

- Búsqueda con retroceso: los únicos nodos vivos son los que están en el camino de la raíz al nodo en expansión
- Ramificación y poda: puede haber más nodos vivos. Se necesita una estructura de datos auxiliar para almacenar **la lista de nodos vivos**

Recorrido del árbol de soluciones

Diferentes estrategias de elegir el siguiente nodo de la lista de nodos vivos

$\Downarrow$

Distintos órdenes de recorrido del árbol de soluciones

- **FIFO**: la lista de nodos vivos es una cola $\Rightarrow$ recorrido por niveles (en anchura)
- **LIFO**: la lista de nodos vivos es una pila $\Rightarrow \approx$ recorrido en profundidad (*D-search*)

- **Mínimo coste**: la lista de nodos vivos es una cola con prioridades $\Rightarrow$ recorrido extraño
- La prioridad de un nodo se calcula con una **función de estimación** que mide cuánto de *prometedor* es un nodo

Punto clave: función de estimación

Encontrar un buen orden de recorrido (o ramificación) de los nodos

$$\Downarrow$$

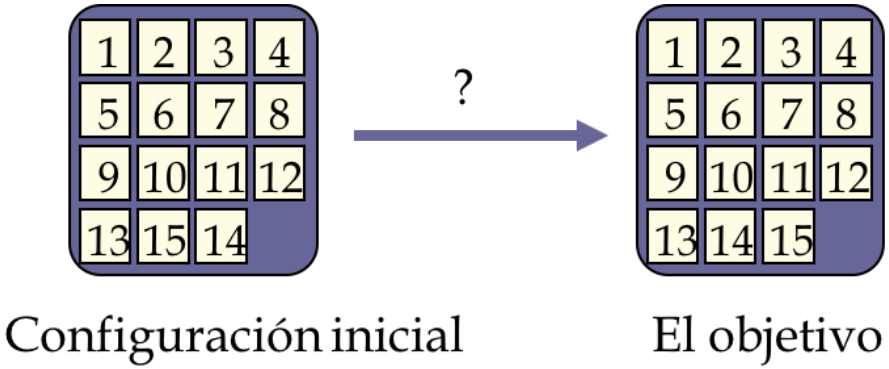
Definir una buena función de prioridad de los nodos vivos, para que las soluciones buenas se encuentren rápidamente

Esquema algorítmico


```
1  repetir
2   elegir el nodo más prometedor como nodo_E;
3   generar todos sus hijos;
4   matar el nodo_E;
5   para cada hijo hacer
6     si coste(hijo) > coste(mejor_solución_en_curso) entonces se mata
7     sino
8       si no es solución entonces se pasa a la lista_de_nodos_vivos
9       sino {es solución: el coste no es estimado sino real}
10        es la mejor_solución_en_curso y se revisa la lista_de_nodos_vivos,
11        eliminando los que prometen algo peor
12      fsi
13    fsi
14  fpara
15 hasta que la lista está vacía
```

El juego del 15

- El problema original



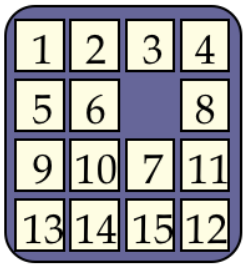
Samuel Loyd and the 15 puzzle

`http://mathematicalmysterytour.blogspot.com/2014/10/sam-loyd-and-15-puzzle.html`

- La solución de fuerza bruta:
  - Buscar en el espacio de estados (todas las configuraciones alcanzables dede la configuración inicial) hasta encontrar el objetivo
  - Hay $16! \approx 20,9 \times 10^{12}$ configuraciones, aunque **sólo (?) la midad pueden alcanzarse** desde la configuración inicial propuesta por Loyd

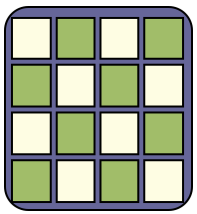
Configuraciones alcanzables

- Numeremos las casillas de 1 a 16
- Dada una configuración:
  - Sea $pos(i)$ la posición (entre 1 y 15) de la ficha con número i, y sea $pos(16)$ la posición de la casilla vacía
  - Para cada ficha i, sea $m(i)$ el número de fichas $j < i$ tales que $pos(j) > pos(i)$



$m(i)$=0, i=1,2,3,4,5,6,7
$m(8)$=$m(9)$=$m(10)$=1
$m(11)$=0
$m(12)$=0
$m(13)$=$m(14)$=$m(15)$=1
$m(16)$=9

  - Sea $x = 1$ si la casilla vacía está en alguna de las posiciones verdes y $x = 0$ en caso contrario



Configuraciones alcanzables

Teorema

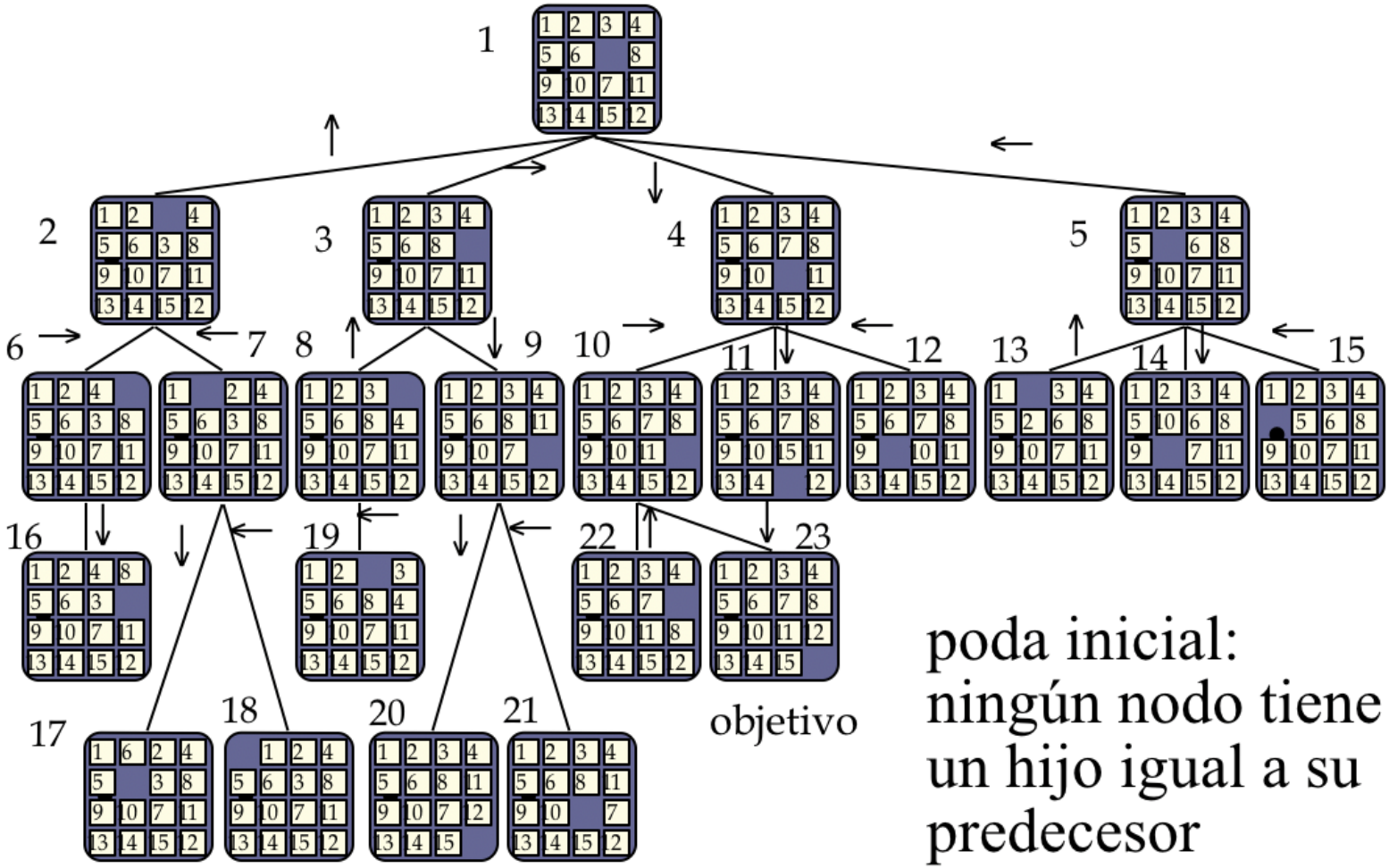
La configuración objetivo es alcanzable desde una cierta configuración inicial si para esta configuración:

$$\sum_{i=1}^{16} m(i) + x \quad \textit{es par}.$$

- Para el ejemplo anterior, dicha suma vale 16: la configuración objetivo es alcanzable
- En el problema original de Loyd, la configuración objetivo no es alcanzable

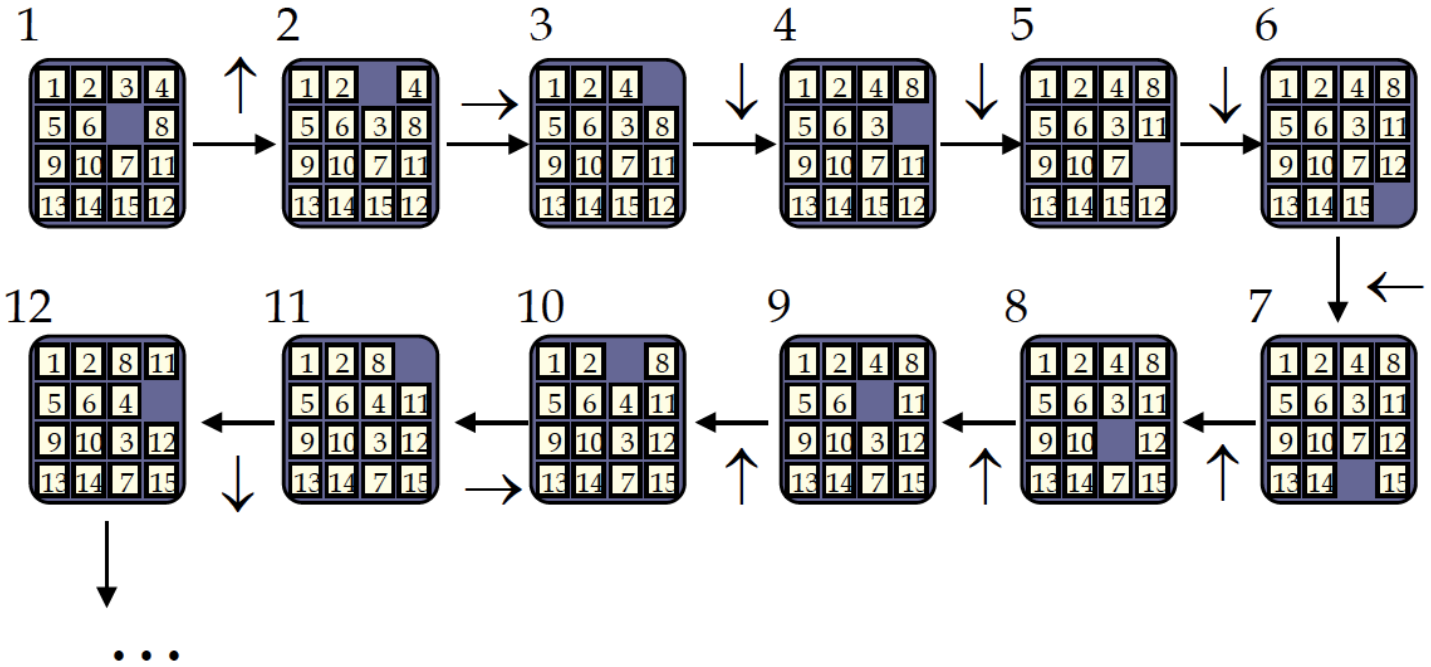
Búsqueda en anchura

- Hijos de cada nodo = movimientos de la casilla vacía (arriba,dcha,abajo,izda)



Búsqueda en profundidad

- Orden de los movimientos (arriba,dcha,abajo,izda)



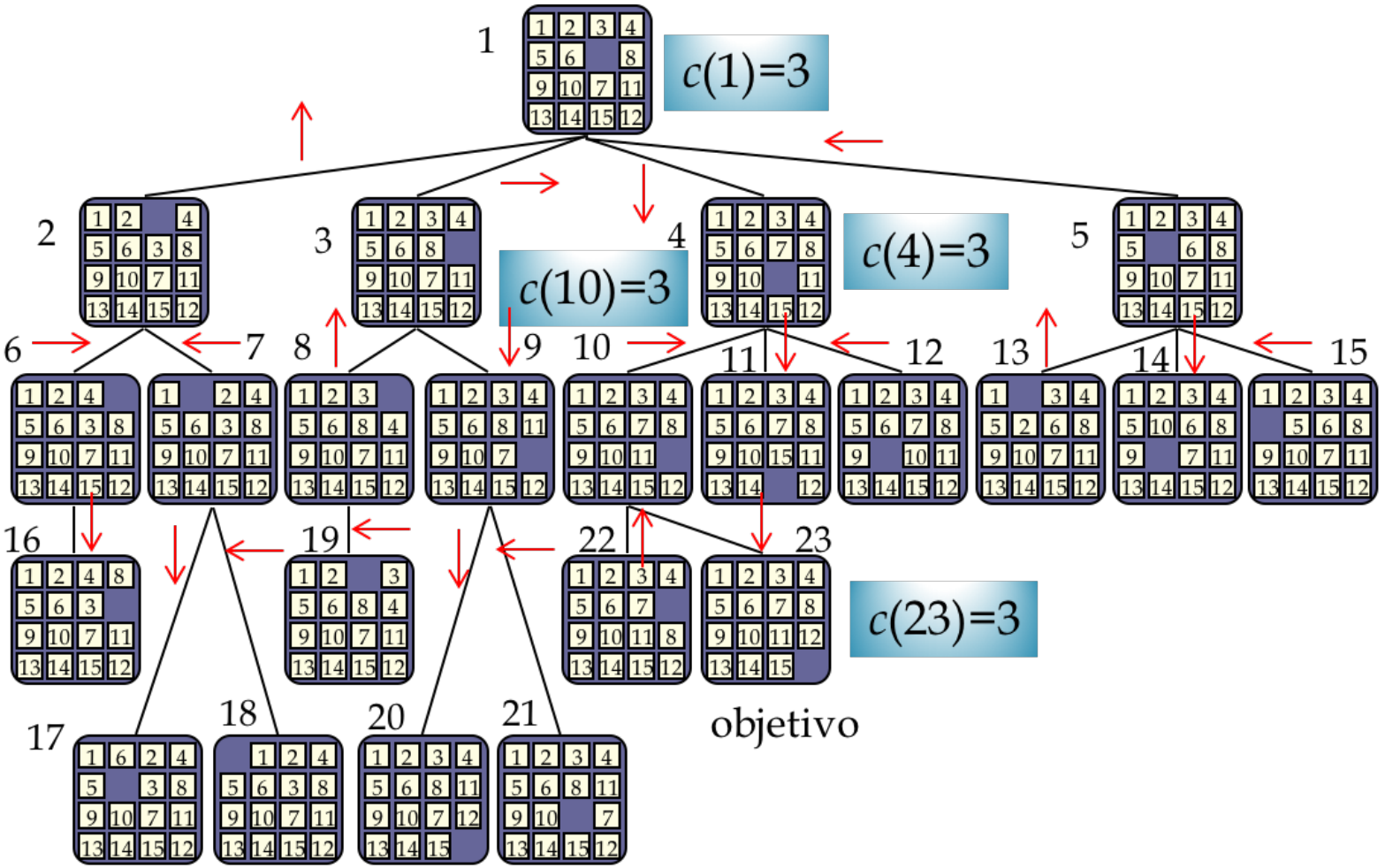
  - Sigue la rama izquierda del árbol

Estrategias de búsqueda

- Estrategias a **ciegas**:
  - FIFO: recorrido en anchura o por niveles
    - Relativamente útil si el nodo solución está cerca de la raíz
  - LIFO: $\approx$ recorrido en profundidad (también llamado *D-search* en el libro de Horowitz et al.)
    - Puede no encontrar la solución (problemas con los ciclos)
- Estrategias no ciegas (*informadas*, en inteligencia artificial)
  - Buscan la solución adaptando el camino de búsqueda en el árbol de soluciones a la instancia del problema a solucionar
  - Se necesita definir una **función de coste** para los nodos del árbol

Ejemplo de función de coste

$c(x)$ = longitud desde la raíz hasta el nodo objetivo más cercano que es descendiente de x



Función de estimación

- Problema: el cálculo de la función c
  - Un ejemplo de problemas en los que es posible: aquéllos en los que hay soluciones voraces

Función de estimación (heurística)

$$\hat{c}(x) = f(x) + \hat{g}(x)$$

$f(x)$ = longitud del camino de la raíz a x
$\hat{g}(x)$ = estimación de la longitud del camino de x hasta la solución más cercana

Propiedades de $\hat{c}$:

- Fácil de calcular
- Si x es una hoja o solución: $c(x) = \hat{c}(x)$

Estrategia de mínimo coste

$\hat{g}(x)$ = número de casillas no vacías que no están en su sitio objetivo ($\hat{c}(x) \leq c(x), \forall x$)

- La lista de nodos vivos es una cola de nodos x con prioridades $\hat{c}(x)$
- Nodo inicial

1

$\hat{c}(1) = 0 + 3$

- Se generan sus hijos y se añaden a la cola

2

$\hat{c}(2) = 1 + 4$

3

$\hat{c}(3) = 1 + 4$

4

$\hat{c}(4) = 1 + 2$

5

$\hat{c}(5) = 1 + 4$

Estrategia de mínimo coste

- Se elige el mínimo (4), se elimina de la cola y se generan sus hijos

2  $\hat{c}(2) = 1 + 4$

3  $\hat{c}(3) = 1 + 4$

5  $\hat{c}(5) = 1 + 4$

10  $\hat{c}(10) = 2 + 1$

11  $\hat{c}(11) = 2 + 3$

12  $\hat{c}(12) = 2 + 3$

- Se elige el mínimo (10), se elimina de la cola y se generan sus hijos

2  $\hat{c}(2) = 1 + 4$

3  $\hat{c}(3) = 1 + 4$

5  $\hat{c}(5) = 1 + 4$

11  $\hat{c}(11) = 2 + 3$

12  $\hat{c}(12) = 2 + 3$

22  $\hat{c}(22) = 3 + 2$

23  $\hat{c}(23) = 3 + 0$

Algoritmo de búsqueda

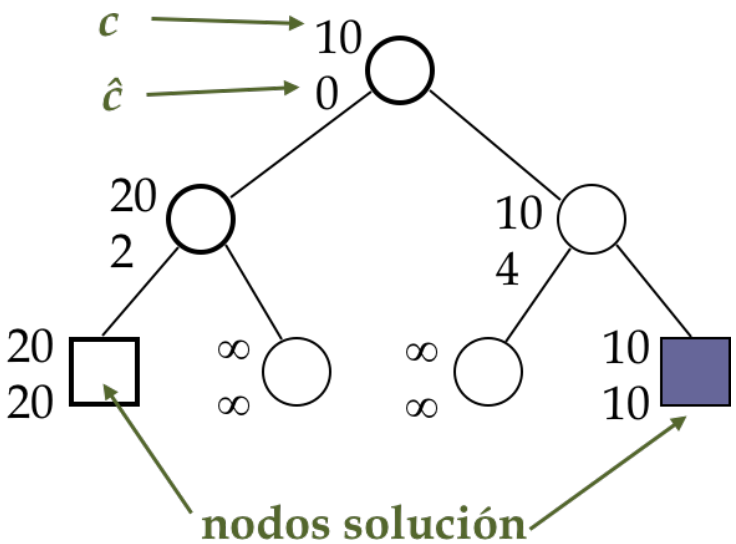

```
1  algoritmo mínimoCoste(ent x0: nodo)
2  variables c:cola; {cola con prioridades <x,coste(x)>}
3            éxito: bool; xcurso,x:nodo
4  principio
5    si esSolución(x0) entonces escribir(x0)
6    sino
7      crearColaVacía(c); {cola de nodos vivos}
8      añadir(c,<x0,coste(x0)>); éxito:=falso;
9      mientrasQue not éxito and not esVacía(c) hacer
10       xcurso:=min(c); {nodo en expansión}
11       eliminar(c,xcurso);
12       mientrasQue not éxito and hay_otro_hijo_x_de_xcurso hacer
13         si esSolución(x) entonces escribir(x); éxito:=verdadero
14         sino añadir(c,<x,coste(x)>) fsi
15       fmientrasQue
16     fmientrasQue
17     si not éxito entonces escribir("No hay solución") fsi
18   fsi
19 fin
```

Observaciones

- Si se desea escribir todo el camino desde la raíz hasta la solución, cada vez que se añade un nodo a la cola, hay que guardar en una **tabla auxiliar de padres de nodos** ese nodo junto a su padre
- La terminación del algoritmo sólo está garantizada si el espacio de estados es finito
- Si el espacio de estados es infinito y existe al menos una solución, se puede garantizar la terminación con una elección adecuada de la función de estimación
- Si el espacio de estados es infinito y no hay solución, el algoritmo no termina...
  - Se suele restringir la búsqueda a nodos con coste estimado menor que C (cota)

Problemas de optimización

- La función o coste a minimizar ya no es la distancia de la raíz a la solución, sino una función objetivo dada
- ¿Qué ocurre si interesa buscar una solución óptima (es decir de coste mínimo)?
- ¿Encuentra el algoritmo `minimoCoste` esa solución? **No**



- Se expande la raíz, sus hijos se añaden a la cola de nodos vivos
- Se elige el hijo izquierdo, se expande y se obtiene una solución factible de coste 20
- La solución óptima tiene coste 10.
- ¿Porqué?

$$\exists x, y : \hat{c}(x) < \hat{c}(y) \wedge c(x) > c(y)$$

Problemas de optimización

Teorema

Si para cada nodo x del árbol del espacio de estados $\hat{c}(x)$ es una estimación de $c(x)$ tal que para todo par de nodos y, z:

$$\hat{c}(y) < \hat{c}(z) \Leftrightarrow c(y) < c(z)$$

entonces el algoritmo `mínimoCoste` termina y encuentra siempre una solución óptima.

- **Problema:** no es fácil encontrar una función de estimación con tales características y fácil de calcular
- **Conformarse con:** una función $\hat{c}(x)$ fácil de calcular y tal que para todo nodo x: $\hat{c}(x) \leq c(x)$ y para cada solución: $\hat{c}(x) = c(x)$. En este caso, `mínimoCoste` no siempre encuentra una solución óptima, pero se puede modificar ligeramente...

Algoritmo de búsqueda modificado


```
1  algoritmo mínimoCoste2(ent x0: nodo)
2  variables c:cola; {cola con prioridades <x,coste(x)>}
3            éxito: bool; xcurso,x:nodo
4  principio
5      crearColaVacía(c); {cola de nodos vivos}
6      añadir(c,<x0,coste(x0)>); éxito:=falso;
7      mientrasQue not éxito and not esVacía(c) hacer
8        xcurso:=min(c); {nodo en expansión}
9        eliminar(c,xcurso);
10       si esSolución(xcurso) entonces escribir(x); éxito:=verdadero
11       sino
12         para todo x_hijo_de_xcurso hacer
13           añadir(c,<x,coste(x)>)
14         fpara
15       fsi
16     fmientrasQue
17     si not éxito entonces escribir("No hay solución") fsi
18 fin
```

Diferencia entre los dos algoritmo de coste mínimo

- `mínimoCoste` si encuentra un hijo que sí es solución factible ya no incluye el resto de los hijos en la cola de nodos vivos con prioridades;
- `mínimoCoste2` incluye en la cola de nodos vivos con prioridades todos los hijos, sin mirar si son solución factible o no. ($\Leftarrow$ algoritmo más prudente)

Teorema

Si para cada nodo x del árbol del espacio de estados $\hat{c}(x)$ es una función de estimación tal que:

$$\hat{c}(x) \leq c(x)$$

y para cada nodo solución x se verifica: $\hat{c}(x) = c(x)$, entonces si el algoritmo `mínimoCoste2` encuentra una solución, ésta es óptima.

Demostración teorema

- Si se encuentra la solución `xcurso`, se tiene que: $\hat{c}(xcurso) \leq \hat{c}(x)$ para todo nodo x de la cola de nodos vivos
- Además: $\hat{c}(xcurso) = c(xcurso)$ y $\hat{c}(x) \leq c(x)$ para todo nodo x de la cola de nodos vivos y
- También $c(x) \leq c(y)$ para cada nodo x del árbol, donde y es un hijo de x
- Luego $c(xcurso) \leq c(x)$ y `xcurso` es de mínimo coste.

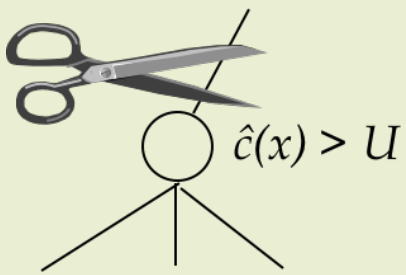
Poda

- Dado el problema de minimización de una **función objetivo** $c(x)$
- Se utiliza una función de estimación $\hat{c}(x)$ tal que $\hat{c}(x) \leq c(x)$
- Suponer que se conoce una cota superior del valor mínimo de c:

$$c(x^*) = \min_x c(x) \leq \mathcal{U}$$

(cota pesimista de la solución del problema)

Regla de poda



- Si $\hat{c}(x) > \mathcal{U}$, entonces x puede ser podado porque $\forall y$ solución descendiente de x:

$$c(y) \geq c(x) \geq \hat{c}(x) > \mathcal{U}$$

Poda

- Valor inicial de $\mathcal{U}$:
  - Utilizar alguna heurística (basada en información extra sobre el problema)
  - ∞
- Si al alcanzar solución factible x: $c(x) < \mathcal{U}$, se actualiza el valor de $\mathcal{U}$
- Todos los nodos vivos de coste estimado mayor que $\mathcal{U}$ también se pueden podar
- Si el valor inicial de $\mathcal{U}$ es mayor o igual que el coste de una solución de mínimo coste, la regla de poda no elimina ningún nodo ascendente de una solución de coste mínimo

Construcción del algoritmo

Definición de la función de coste $c(x)$ de forma que $c(x)$ sea mínimo para todos los nodos que representan una solución óptima. ¿Cómo?

- Si x es solución factible del problema: $c(x) = F_{objectivo}(x)$
- Si x no es factible: $c(x) = \infty$
- Si x es solución parcial (x puede ser completada a factible):

$$c(x) = min\{c(y) \mid y \text{ es descendiente de } x\}$$

Tan difícil de calcular como resolver el problema original!

- Se utiliza $\hat{c}(x)$ tal que $\hat{c}(x) \leq c(x)$ para todo x
- La función de estimación **estima el valor de la función objetivo y no el coste computacional** de alcanzar la solución

Puntos clave de método

1. Encontrar un buen orden de recorrido (o ramificación) de los nodos, es decir, definir una buena función de prioridad $\hat{c}$ de los nodos vivos, para que las soluciones buenas se encuentren rápidamente.
2. **Encontrar una buena función de poda, $\mathcal{U}$ para que se produzca el retroceso lo antes posible.**

Algoritmo de búsqueda modificado


```
1  {Pre: Existe al menos un nodo de mínimo coste, existe una función c_est
2        y un valor inicial U: c_est(x)<=c(x)<=U.
3   Post: Encuentra en el árbol cuya raíz es x0 una solución de mínimo coste}
4  algoritmo RamPodaMinCoste(ent x0: nodo)
5  variables q:cola; {cola con prioridades de <x,c_est(x)>} xcurso,x: nodo
6  principio
7   creaColaVacía(q); {cola con prioridades de nodos vivos}
8   añadir(q,<x0,c_est(x0)>);
9   mientrasQue not esVacía(q) hacer
10    xcurso:= min(q); {nodo en expansión} eliminar(q,xcurso);
11    para todo x hijo_de xcurso hacer
12      si c_est(x) <= U entonces
13        añadir(q,<x,c_est(x)>);
14        si esSoluciónFactible(x) and c(x) < U entonces U:=c(x) fsi
15      fsi
16    fpara;
17    si esVacía(q) or c_est(min(q))>=U entonces escribir(U); crearVacía(q) fsi
18   fmientrasQue
19 fin
```

Observaciones

- Si se desea escribir la solución óptima además del valor óptimo U de la función objetivo:
  - Cada vez que se añade un nodo a la cola, hay que guardar en una tabla auxiliar de nodos padre el nodo añadido junto a su padre o bien **guardar con cada nodo una identificación de su padre**, y
  - Hay que mantener una variable auxiliar con el valor del último nodo solución que ha permitido actualizar U

Un problema de planificación de tareas a plazo fijo

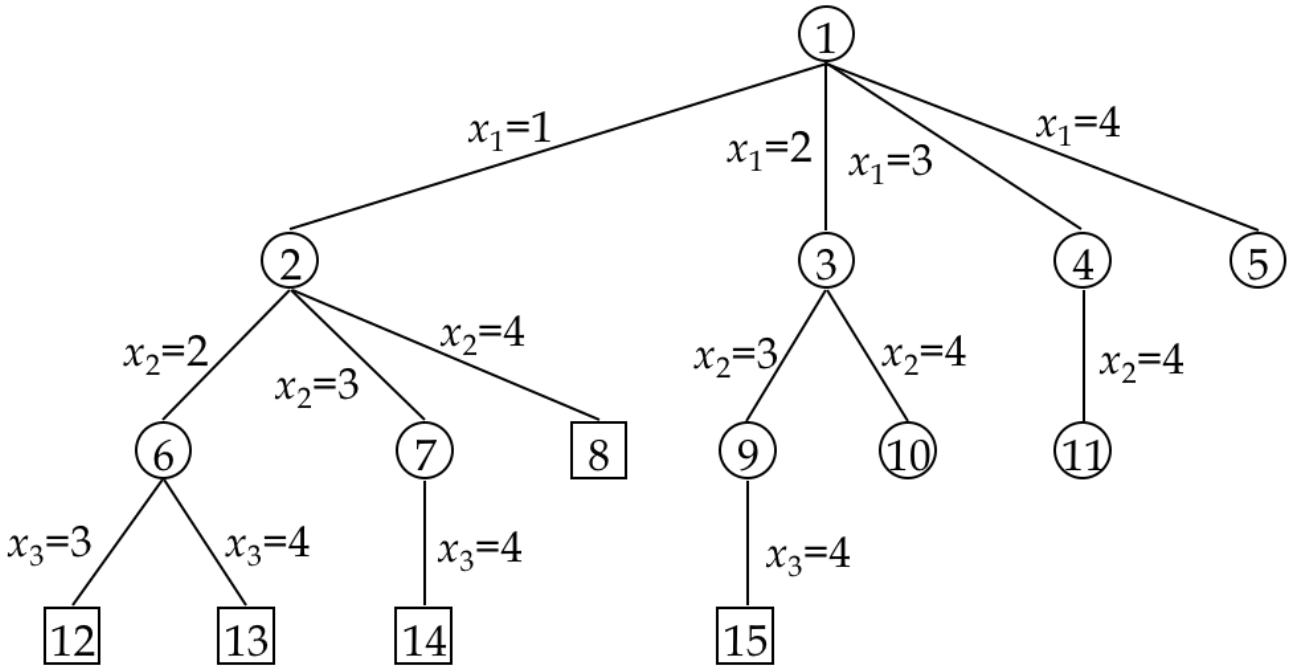
- Se tiene un conjunto $C = \{1, 2, \ldots, n\}$ de tareas y en cada instante sólo se puede realizar una tarea
- Cada tarea, $1 \leq i \leq n$, tiene asociada una terna (t_i, d_i, w_i)
  - t_i es la **duración** de la tarea i
  - la tarea i debe terminarse antes del **plazo** d_i
  - hay una **penalización** w_i si la tarea i no termina antes de su plazo
- Se quiere encontrar un subconjunto $S \subseteq C$ tal que todas las tareas de S puedan ser ejecutadas antes de sus plazos y la penalización total sea mínima.

Ejemplo

- $n = 4$
- $(t_1, d_1, w_1) = (1, 1, 5)$
- $(t_2, d_2, w_2) = (2, 3, 10)$
- $(t_3, d_3, w_3) = (1, 2, 6)$
- $(t_4, d_4, w_4) = (1, 1, 3)$

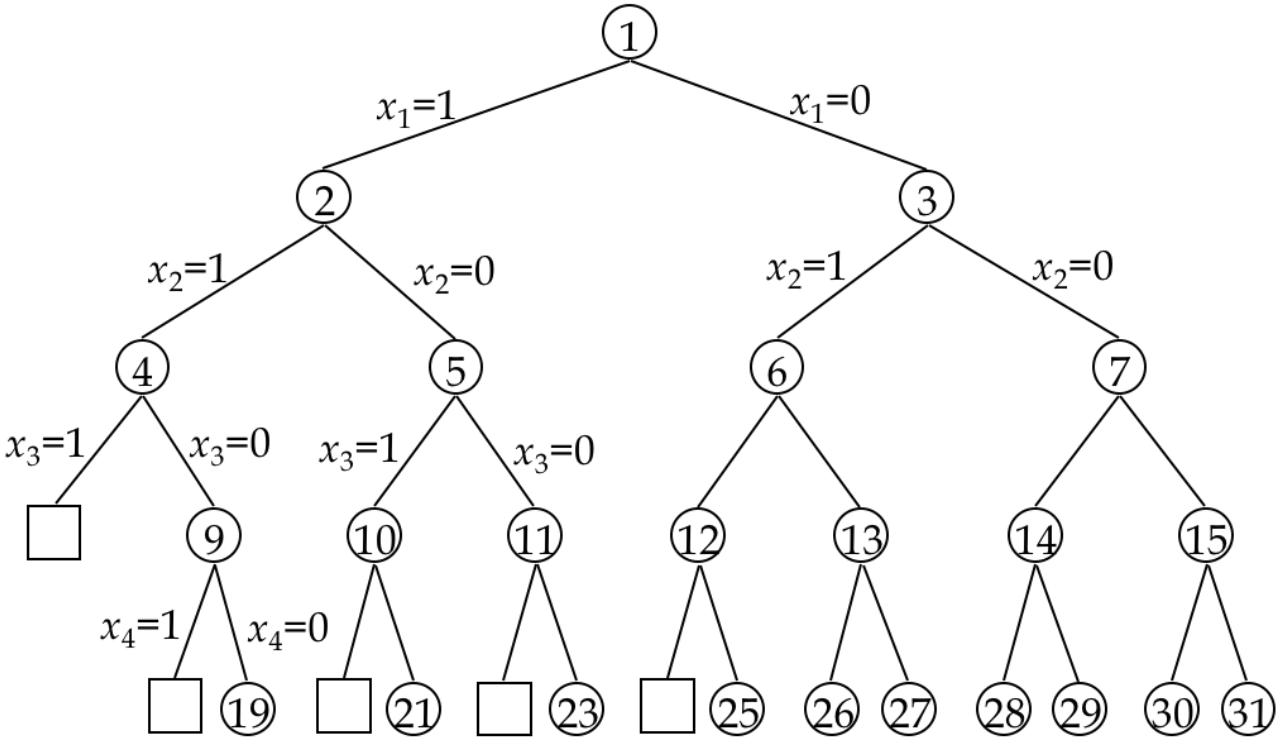
- Espacio de estados: todos los subconjuntos de $\{1, 2, 3, 4\}$
- Dos representaciones posibles:
  - Tuplas de tamaño variable
  - Tuplas de tamaño fijo

Representación con tuplas de tamaño variable



- Nodos cuadrados: estados no factibles
- Nodos circulares: soluciones factibles
- Nodo 9: solución óptima (es la única):
  - $S = \{2, 3\}$; penalización total =8

Representación con tuplas de tamaño fijo

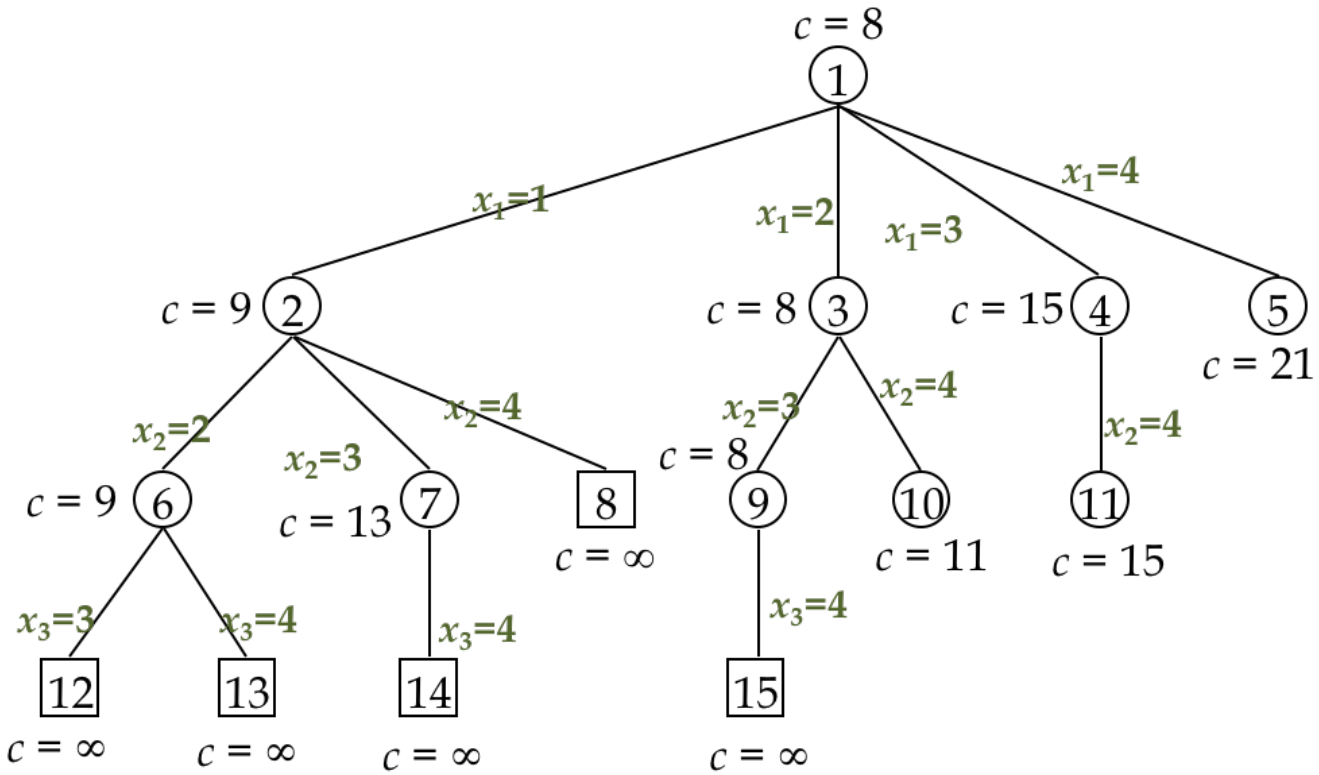


- Nodos cuadrados: estados no factibles
- Hojas circulares: soluciones factibles
- Nodo 25 solución óptima: $S = \{2, 3\}$; penalización total =8

Definición de la función de coste $c(x)$

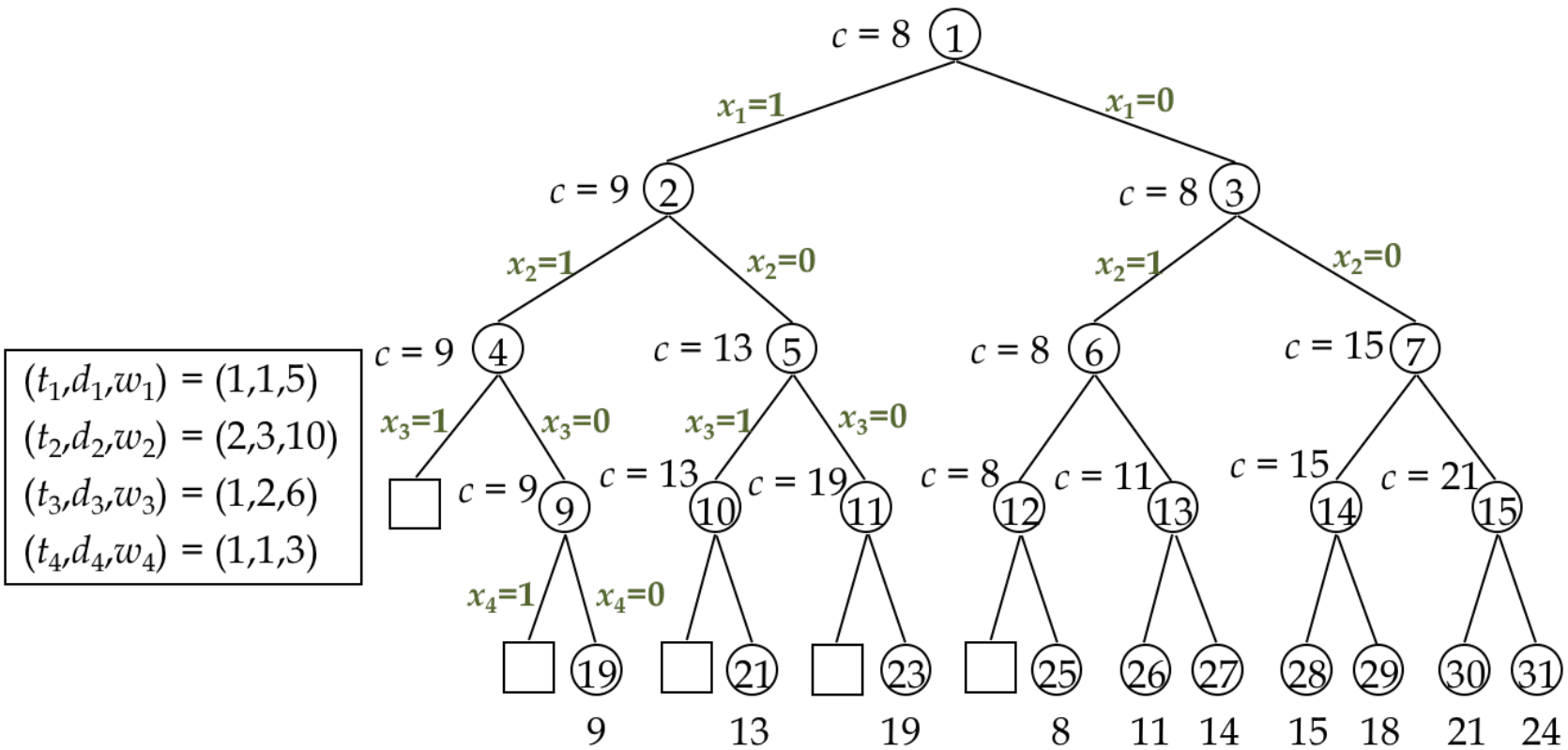
Para todo nodo circular x con algún hijo circular $c(x)$ es la mínima penalización correspondiente a todos los nodos descendientes de x. Para todo nodo cuadrado x: $c(x) = \infty$.

$(t_1,d_1,w_1) = (1,1,5)$
$(t_2,d_2,w_2) = (2,3,10)$
$(t_3,d_3,w_3) = (1,2,6)$
$(t_4,d_4,w_4) = (1,1,3)$



Definición de la función de coste $c(x)$

Para todo nodo circular x con algún hijo circular $c(x)$ es la mínima penalización correspondiente a todos los nodos descendientes de x. Para todo nodo cuadrado x: $c(x) = \infty$.



Definición de la función de estimación $\hat{c}(x) \leq c(x)$

Sean:

- $S_x = \{\text{tareas seleccionadas para } S \text{ hasta el nodo } x\}$
- $m_x = max\{i \mid i \in S_x\}$

Entonces:

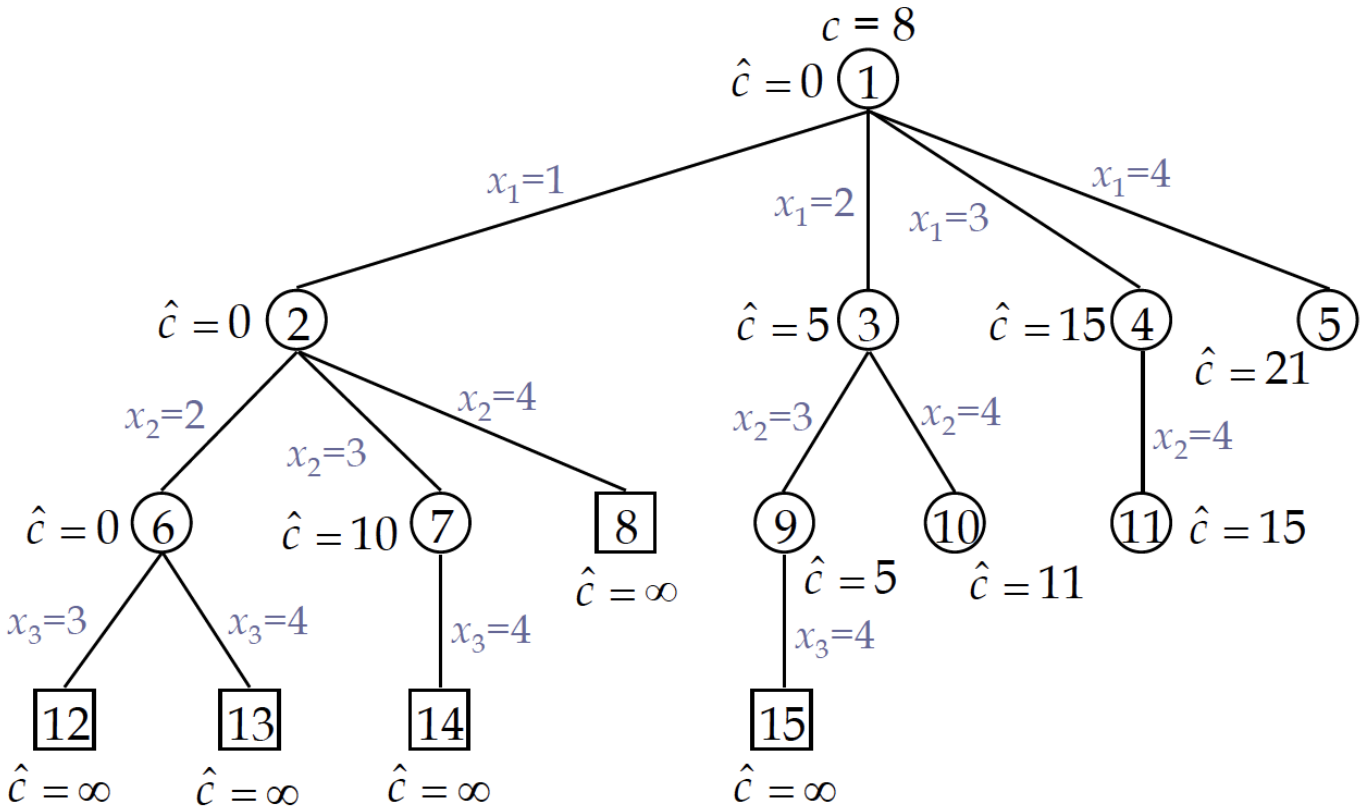
Para los nodos circulares:

$$\hat{c}(x) = \sum_{i<m, i \notin S_x} w_i$$

Penalización debida a las tareas que ya han quedado excluidas de x.
Para los nodos cuadrados:

$$\hat{c}(x) = \infty$$

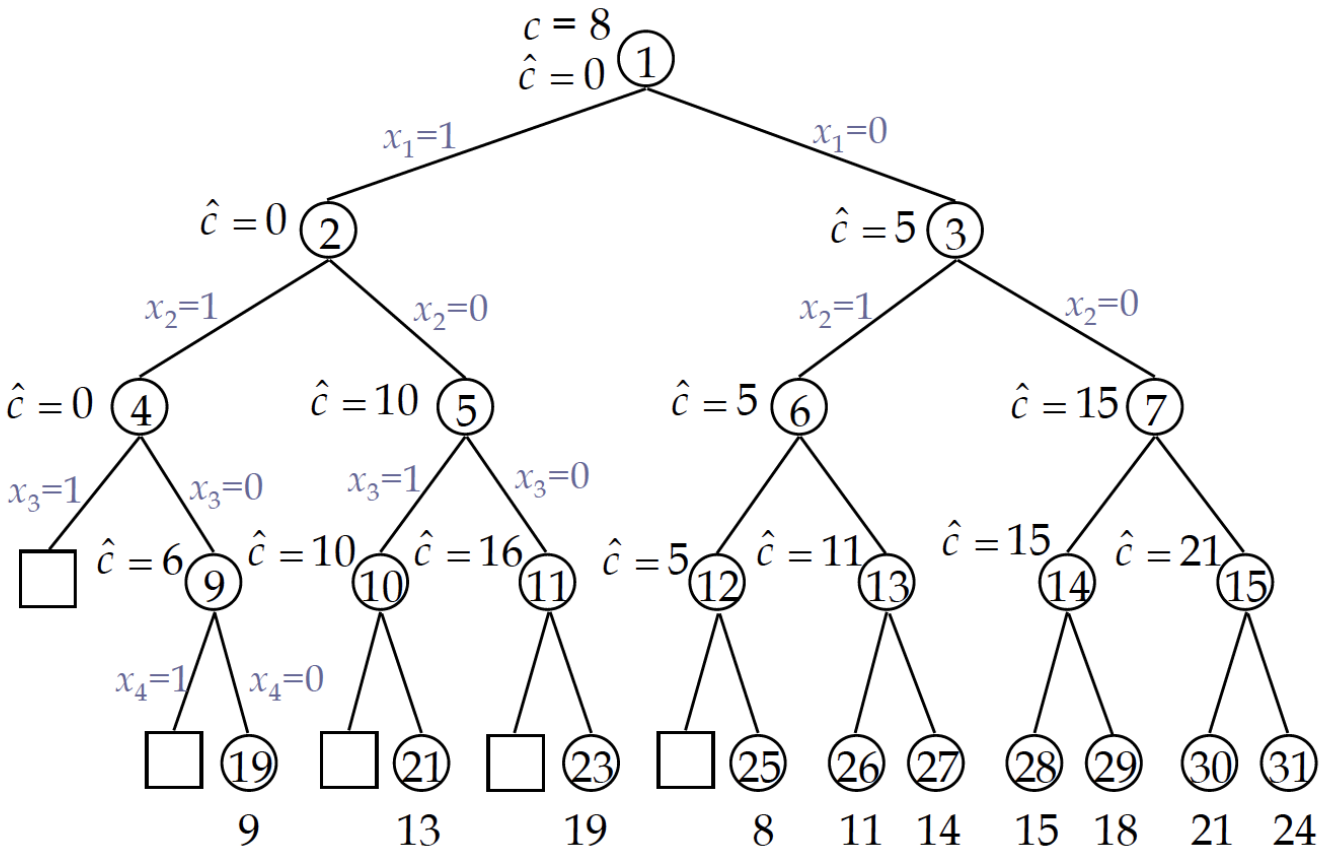
Definición de la función de estimación $\hat{c}(x) \leq c(x)$



$$\hat{c}(x) = \sum_{\substack{i < m_x \\ i \notin S_x}} w_i$$

$(t_1,d_1,w_1) = (1,1,5)$
$(t_2,d_2,w_2) = (2,3,10)$
$(t_3,d_3,w_3) = (1,2,6)$
$(t_4,d_4,w_4) = (1,1,3)$

Definición de la función de estimación $\hat{c}(x) \leq c(x)$



$$\hat{c}(x) = \sum_{\substack{i < m_x \\ i \notin S_x}} w_i$$

$(t_1,d_1,w_1) = (1,1,5)$
$(t_2,d_2,w_2) = (2,3,10)$
$(t_3,d_3,w_3) = (1,2,6)$
$(t_4,d_4,w_4) = (1,1,3)$

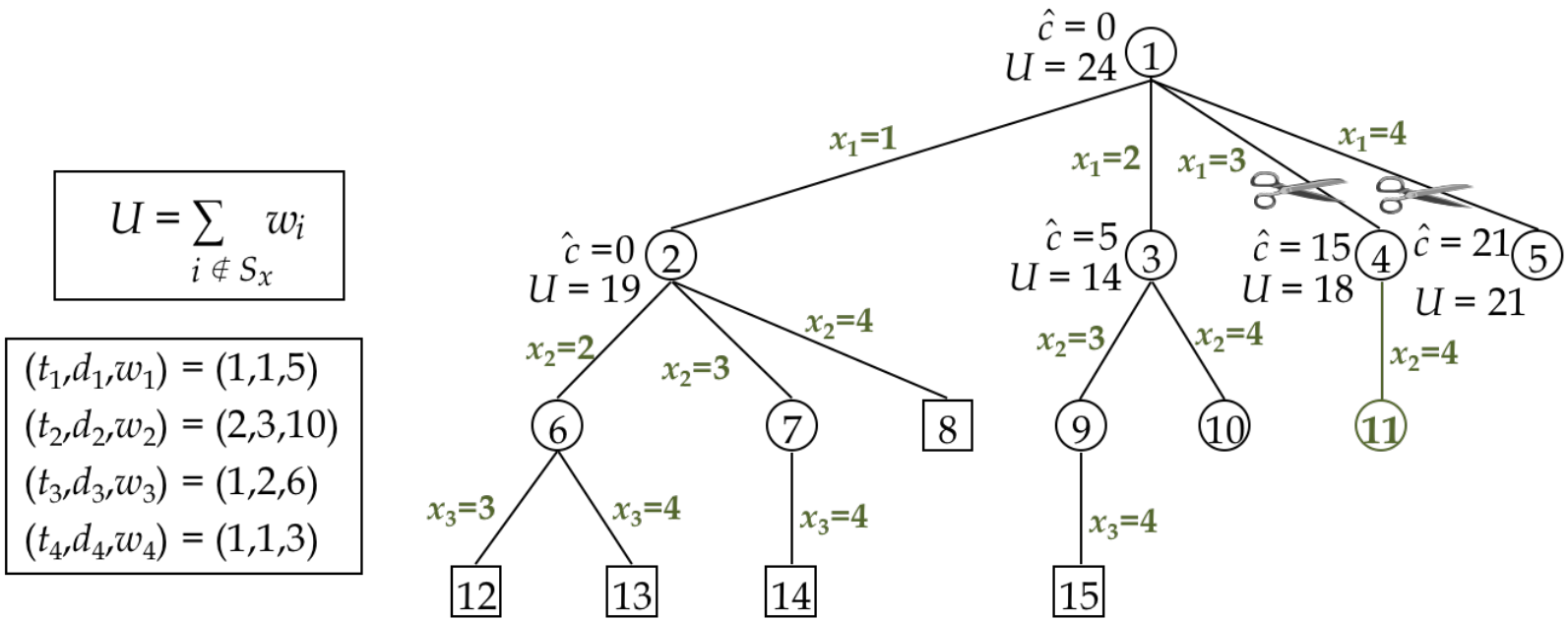
Definición de la función de poda $\mathcal{U}$

- (Recordar) $\mathcal{U}$ debe ser cota superior del coste de una solución de mínimo coste
- Para cada nodo x, se puede calcular un valor de $\mathcal{U}$:

$$\mathcal{U}(x) = \sum_{i \notin S_x} w_i$$

Algoritmo de ramificación y poda

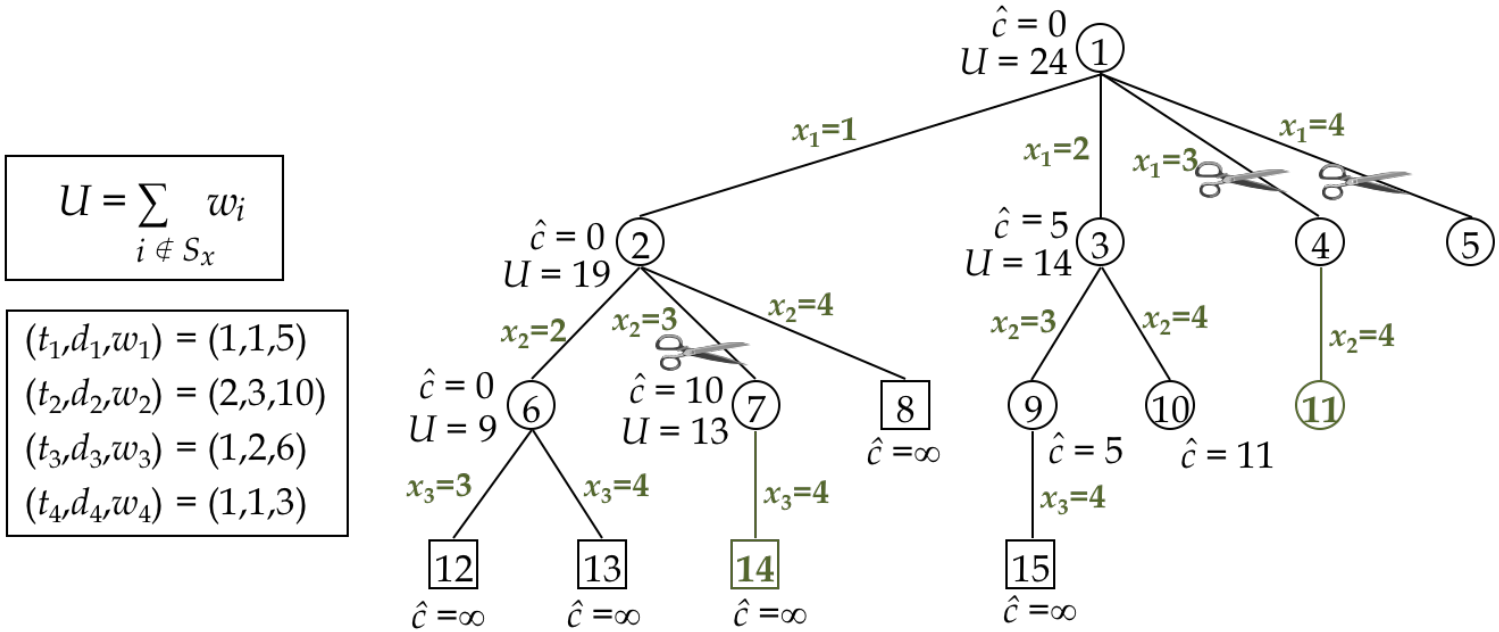
Estrategia de mínimo coste con representación de tamaño variable



- Empieza con $\mathcal{U} = 24$ y (1) como nodo en expansión (único nodo vivo)
- Se expande (1) generando: (2) (3) (4) (5)
- $\mathcal{U}$ se actualiza a 14 al generar (3)
- (4) (5) se matan porque $\hat{c}(4) = 15 > \mathcal{U}$, $\hat{c}(5) = 21 > \mathcal{U}$

Algoritmo de ramificación y poda

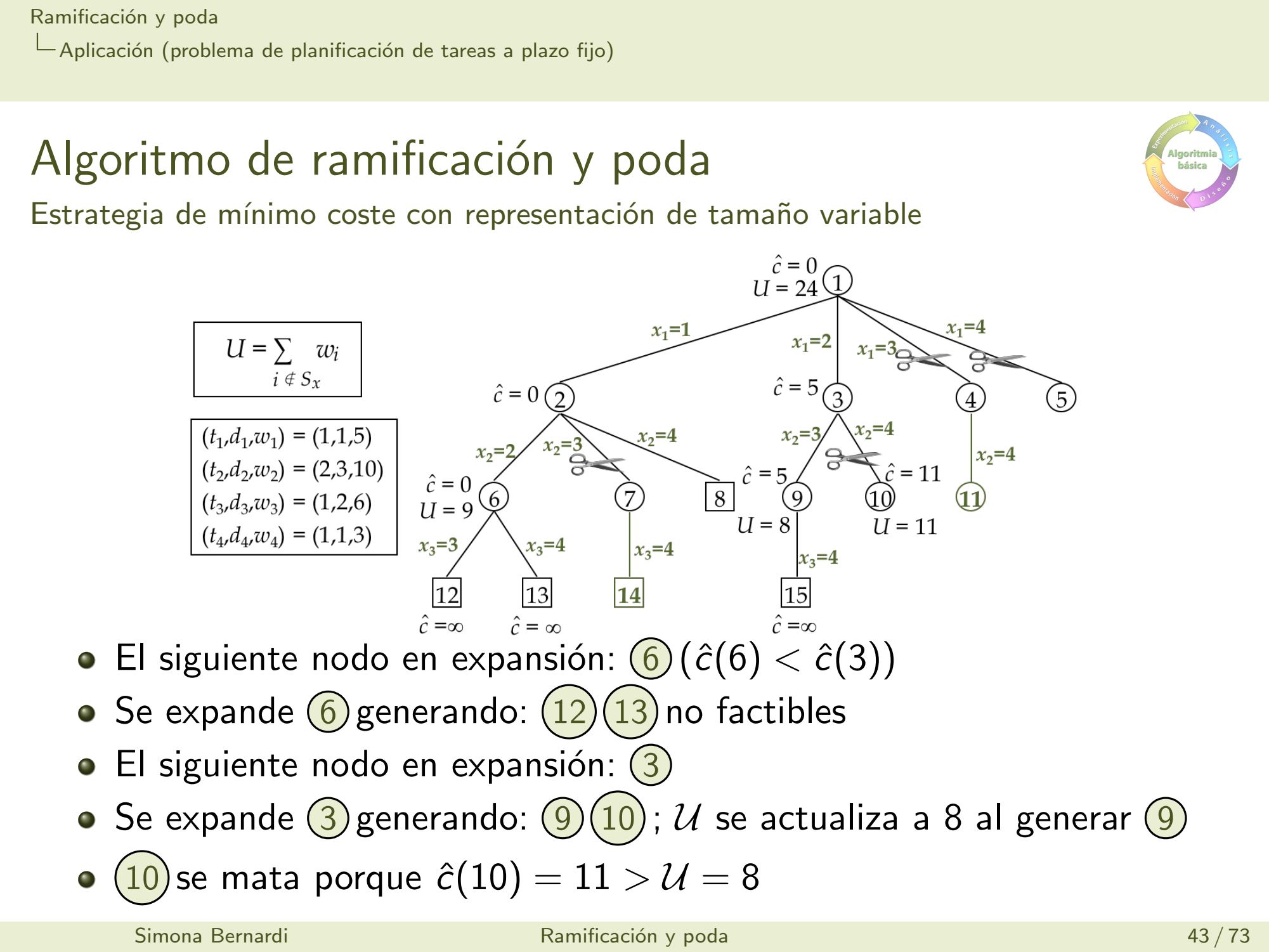
Estrategia de mínimo coste con representación de tamaño variable



- El siguiente nodo en expansión: (2) ($\hat{c}(2) < \hat{c}(3)$)
- Se expande (2) generando: (6) (7) (8); $\mathcal{U}$ se actualiza a 9 al generar (6)
- (7) (8) se matan porque $\hat{c}(7) = 10 > \mathcal{U} = 9$, $\hat{c}(8) = \infty$
- Los nodos vivos son: (3) (6)

Algoritmo de ramificación y poda

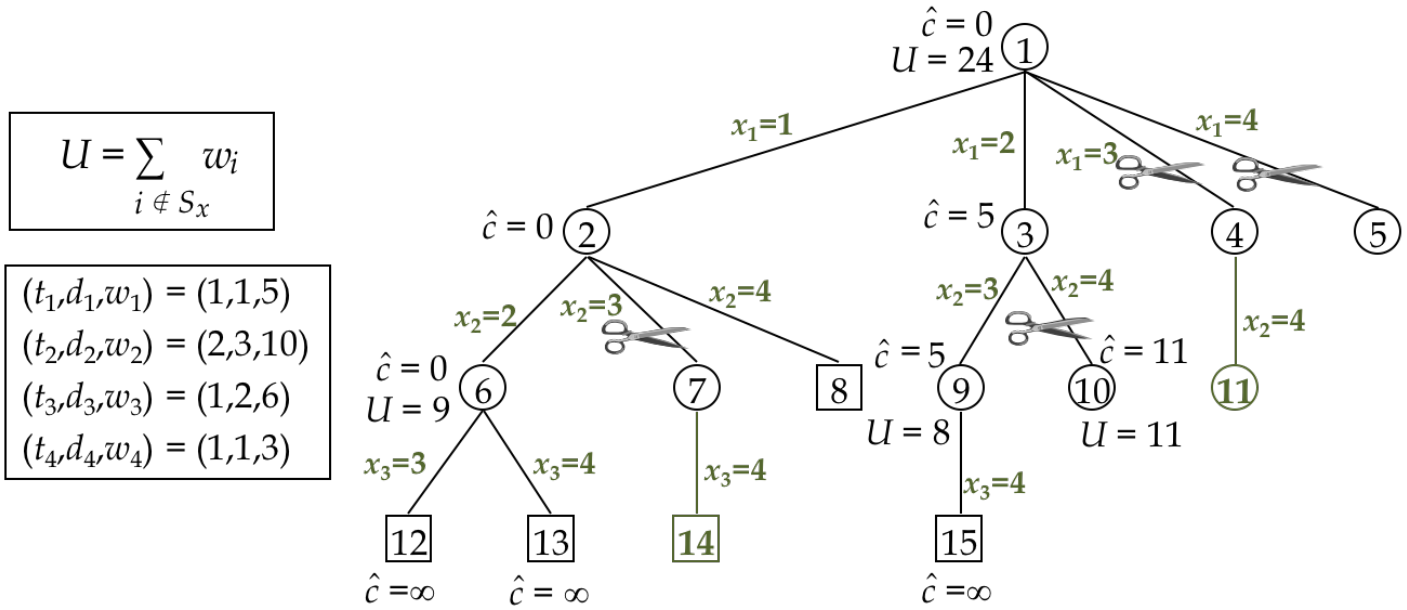
Estrategia de mínimo coste con representación de tamaño variable



- El siguiente nodo en expansión: (6) ($\hat{c}(6) < \hat{c}(3)$)
- Se expande (6) generando: (12) (13) no factibles
- El siguiente nodo en expansión: (3)
- Se expande (3) generando: (9) (10); $\mathcal{U}$ se actualiza a 8 al generar (9)
- (10) se mata porque $\hat{c}(10) = 11 > \mathcal{U} = 8$

Algoritmo de ramificación y poda

Estrategia de mínimo coste con representación de tamaño variable



- El único nodo vivo es (9): se convierte en el nodo en expansión
- Su único hijo es no factible, por tanto se acaba la búsqueda
- La solución es el nodo (9)

Problema de la mochila 0-1

Repaso

- Hay n objetos y una mochila
- El objeto i tiene peso p_i y su inclusión en la mochila produce un beneficio b_i
- **Objetivo:** llenar la mochila de capacidad C, de manera que se maximice el beneficio

$$max \sum_{i=1}^{n} b_i x_i$$

$$s.t. \sum_{i=1}^{n} p_i x_i \leq C$$

$$x_i \in \{0,1\}, b_i, p_i > 0, 1 \leq i \leq n$$

Soluciones a problemas parecidos

- Objetos fraccionables ($0 \leq x_i \leq 1, 1 \leq i \leq n$): **solución voraz**
- Mochila 0-1 con pesos, beneficios y capacidad de la mochila números naturales (**solución de programación dinámica**)
- En el caso general, ninguna de las dos soluciones funciona

Solución con técnica de ramificación y poda

Transformación del problema en un problema de minimización

$$
\begin{aligned}
min \quad & -\sum_{i=1}^{n} b_i x_i \\
s.t. \quad & \sum_{i=1}^{n} p_i x_i \leq C \\
& x_i \in \{0,1\}, b_i, p_i > 0, 1 \leq i \leq n
\end{aligned}
$$

- Espacio de soluciones: 2^n modos de asignar 0 ó 1 a las x_i
- Dos formas de representar la solución (tuplas de tamaño fijo o variable)
  - Elegimos la representación con tuplas de tamaño fijo

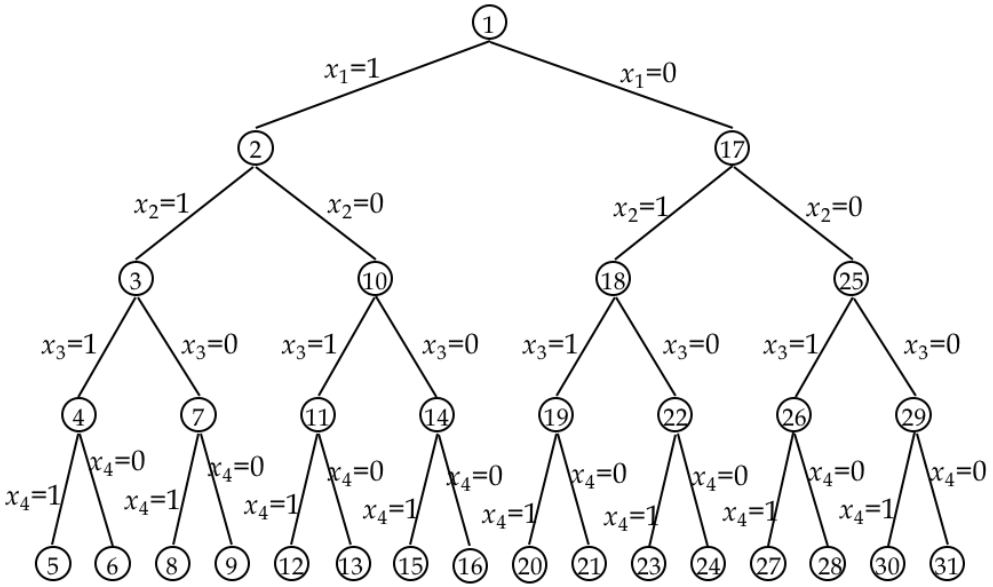
Definición de la función de coste

- Las hojas x tales que:

$$\sum_{i=1}^{n} p_i x_i \leq C$$

representan soluciones **factibles**

- El resto de las hojas representan soluciones **no factibles**



$$c(x) = \begin{cases} -\sum_{i=1}^{n} b_i x_i, & \text{si } x \text{ nodo solución factible} \\ \infty & \text{si } x \text{ nodo solución no factible} \\ min\left(c(x_{izdo}), c(x_{dcho})\right) & \text{si } x \text{ nodo no hoja} \end{cases}$$

Definición de las funciones de estimación y de poda

Funciones $\hat{c}(x)$ y $\mathcal{U}$ tales que:

$$\hat{c}(x) \leq c(x) \leq \mathcal{U}$$

Definición sencilla de $\mathcal{U}$

- Si x es un nodo de nivel j, con $1 \leq j \leq n$, se han asignado ya valores a x_i, $1 \leq i \leq j$, por tanto:

$$c(x) \leq -\sum_{i=1}^{j} b_i x_i = \mathcal{U}$$

- Beneficio actual cambiado de signo

Algoritmo de la función de poda mejorada


```
1  const N=... {número de objetos}
2  tipo vectReal=vector[1..N] de real
3  variables C:real; benef,peso:vectReal
4  {Pre: forall i = 1..n: peso[i]>0 and
5        forall i=1..n-1: benef[i]/peso[i] >= benef[i+1]/peso[i+1]}
6
7  funcion U_mejor(cap,Ufácil:real; obj:entero) devuelve real
8  {cap=capacidad ya ocupada de la mochila; Ufácil=valor de U
9   calculado de forma fácil; obj=índice del primer objeto a considerar}
10 variables: ca,U:real
11 principio
12   U:=Ufácil; ca:=cap;
13   para i:=obj hasta N hacer
14     si ca+peso[i] <= C entonces ca:=ca+peso[i]; U:=U-benef[i] fsi
15   fpara;
16   devuelve U
17 fin
```

$$\mathcal{U} = \texttt{U\_mejor}\left(\sum_{i=1}^{j} p_i x_i, -\sum_{i=1}^{j} b_i x_i, j+1 \right)$$

Definición de la función de estimación

- Utilizar la función de poda que se definió para el método de búsqueda con retroceso
  - Era una cota superior del valor de la mejor solución posible al expandir el nodo y sus descendientes
  - Ahora hemos cambiado de signo la función objetivo
  - Luego, cambiada de signo, es una cota inferior de $c(x)$
- ¿Cómo se calculó esa cota?
  - En el nodo x de nivel j ya se han determinado $x_i, 1 \leq i \leq j$
  - Relajar el requisito de integridad:
    - $x_i \in \{0,1\}, j+i \leq i \leq n$ se sustituye por $0 \leq x_i \leq 1, j+i \leq i \leq n$
  - Aplicar el algoritmo voraz

Algoritmo de la función de estimación


```
1  const N=... {número de objetos}
2  tipo vectReal=vector[1..N] de real
3  variables C:real; benef,peso:vectReal
4  {Pre: forall i = 1..n: peso[i]>0 and
5        forall i=1..n-1: benef[i]/peso[i] >= benef[i+1]/peso[i+1]}
6
7  funcion cota(cap,ben:real; obj:entero) devuelve real
8  {cap=capacidad aún libre de la mochila; ben=beneficio actual;
9   obj=índice del primer objeto a considerar}
10 principio
11  si obj>N or cap=0.0 entonces devuelve ben
12  sino
13    si peso[obj]>cap entonces devuelve ben+cap/peso[obj]*benef[obj]
14    sino
15       devuelve cota(cap-peso[obj],ben+benef[obj],obj+1)
16    fsi
17  fsi
18 fin
```

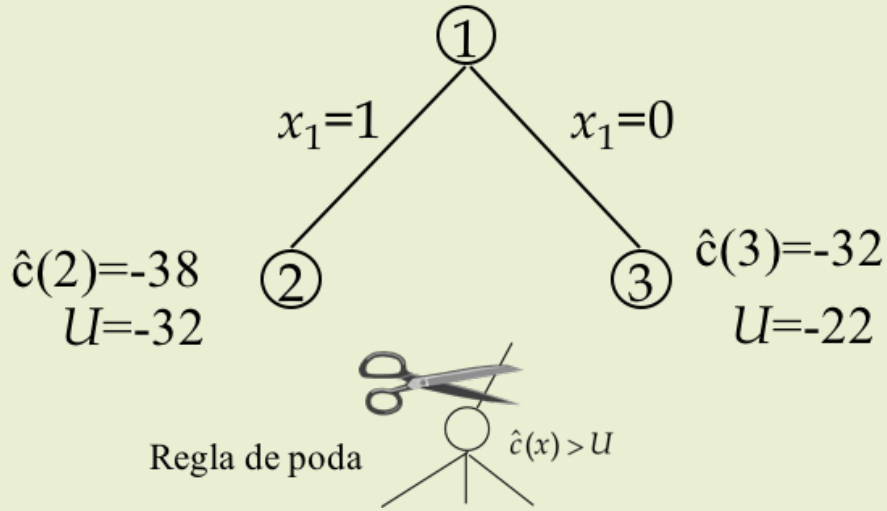
$$\hat{c}(x) = -\text{cota}\left(C - \sum_{i=1}^{j} p_i x_i, \sum_{i=1}^{j} b_i x_i, j+1\right)$$

Ejemplo ../..

- $n = 4$
- $(b_1, b_2, b_3, b_4) = (10, 10, 12, 18)$
- $(p_1, p_2, p_3, p_4) = (2, 4, 6, 9)$
- $C = 15$

- Nodo en expansión: ① (raíz, nivel 0, ninguna x_i asignada)

$$\begin{aligned}
\hat{c}(1) &= -\texttt{cota}(15, 0, 1) \\
&= -\texttt{cota}(13, 10, 2) \\
&= -\texttt{cota}(9, 20, 3) \\
&== -\texttt{cota}(3, 32, 4) \\
&= -(32 + 3/9 * 18) = -38 \\
\mathcal{U} &= -32
\end{aligned}$$



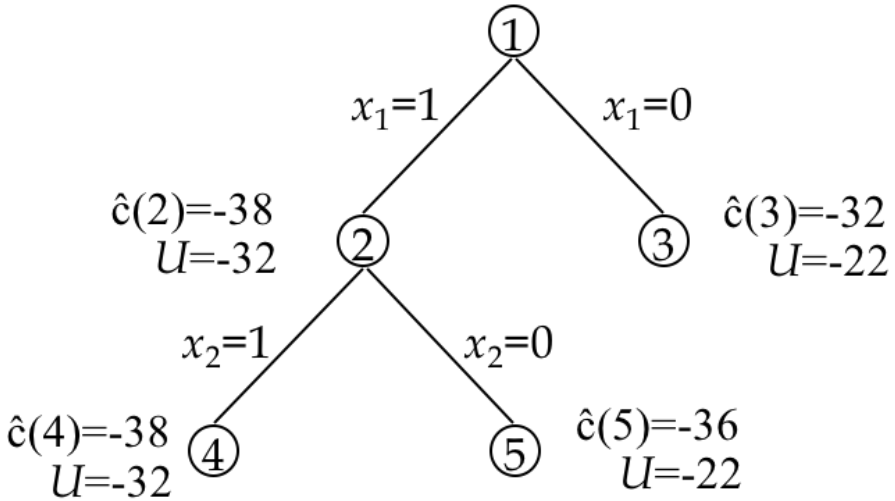
Ejemplo ../..

$n = 4$

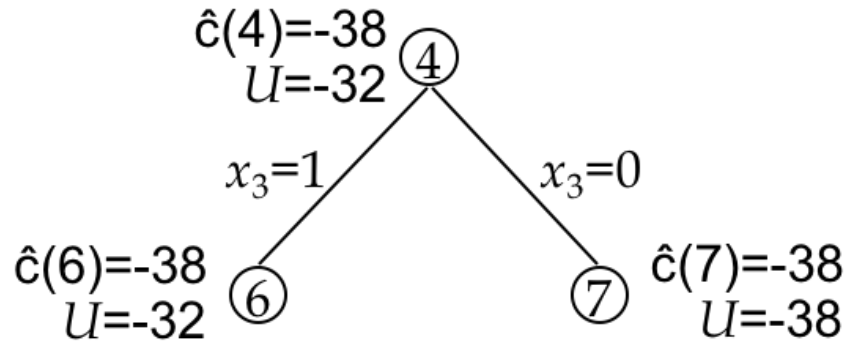
$(b_1, b_2, b_3, b_4) = (10, 10, 12, 18)$

$(p_1, p_2, p_3, p_4) = (2, 4, 6, 9)$

$C = 15$



- (2) siguiente nodo en expansión



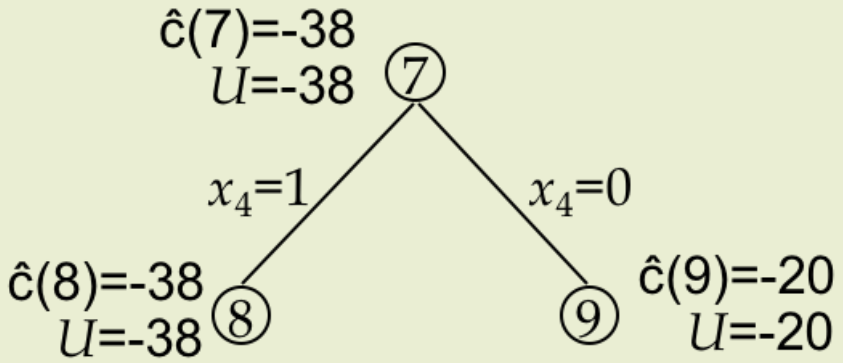
- (4) siguiente nodo en expansión
- ¿Siguiente nodo en expansión?
  Hay (6) y (7) con igual prioridad
  ...

Ejemplo

- Si se elige (6) el hijo izquierdo se elimina por no ser factible, el derecho se añade a la cola de nodos vivos, con coste -32, mayor que el nodo (7)

$$n = 4$$
$$(b_1, b_2, b_3, b_4) = (10, 10, 12, 18)$$
$$(p_1, p_2, p_3, p_4) = (2, 4, 6, 9)$$
$$C = 15$$

- (7) siguiente nodo en expansión
- Nuevo valor de $\mathcal{U} = -38$
- Los nodos (3) (5) y (9) se matan (no alcanzan el valor $\mathcal{U}$)



- El nodo (8) es el único de la cola y es hoja: el valor óptimo es 38

Observaciones

Para poder imprimir la solución (además del valor óptimo) hay que:

- Guardar con cada nodo una identificación de su padre (identificador, puntero, curso,. . .)
- Mantener una variable auxiliar con el valor del último nodo solución que ha permitido actualizar $\mathcal{U}$
- Asociar con cada nodo un booleano que diga si es un hijo izquierdo o derecho (es decir, si se mete en la mochila el objeto correspondiente o no)

Cabe pensar en una versión con estrategia ciega de ramificación (FIFO, por ejemplo):

- Por un lado la intuición nos dice que la estrategia de mínimo coste se acerca más de prisa a la solución óptima
- Por otra parte, la inserción y borrado en la cola con prioridades de nodos vivos tiene coste logarítimico, mientras que con una cola FIFO, el coste es constante . . .

El problema del viajante de comercio

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

- Es decir, dado un grafo dirigido con arcos de longitud no negativa, se trata de encontrar 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 (circuito *hamiltoniano*).

Soluciones propuestas

El problema del viajante de comercio es probablemente el problema NP-completo más estudiado en términos de soluciones propuestas[a]

[a] U. Manber: Introduction to Algorithms. A Creative Approach. Addison Wesley, 1989.

- Solución heurística voraz
  - Calcula una solución subóptima
- Programación dinámica:
  - Coste en tiempo: $\Theta(n^2 2^n)$
  - Coste en espacio: $\Omega(n2^n)$
- El coste del caso peor de la solución que vamos a ver seguirá estando en $\Theta(n^2 2^n)$
  - Sin embargo, el uso de buenas funciones de acotación mejora la eficiencia en muchos casos con respecto a la solución de programación dinámica

Formalización del problema

Sean $G = (V, A)$ un grafo orientado:

- n vértices $V = \{1, 2, \ldots, n\}$
- L_{ij} la longitud de $(i, j) \in A$. Si el arco (i, j) no existe: $L_{ij} = \infty$
- El circuito buscado empieza en el vértice 1

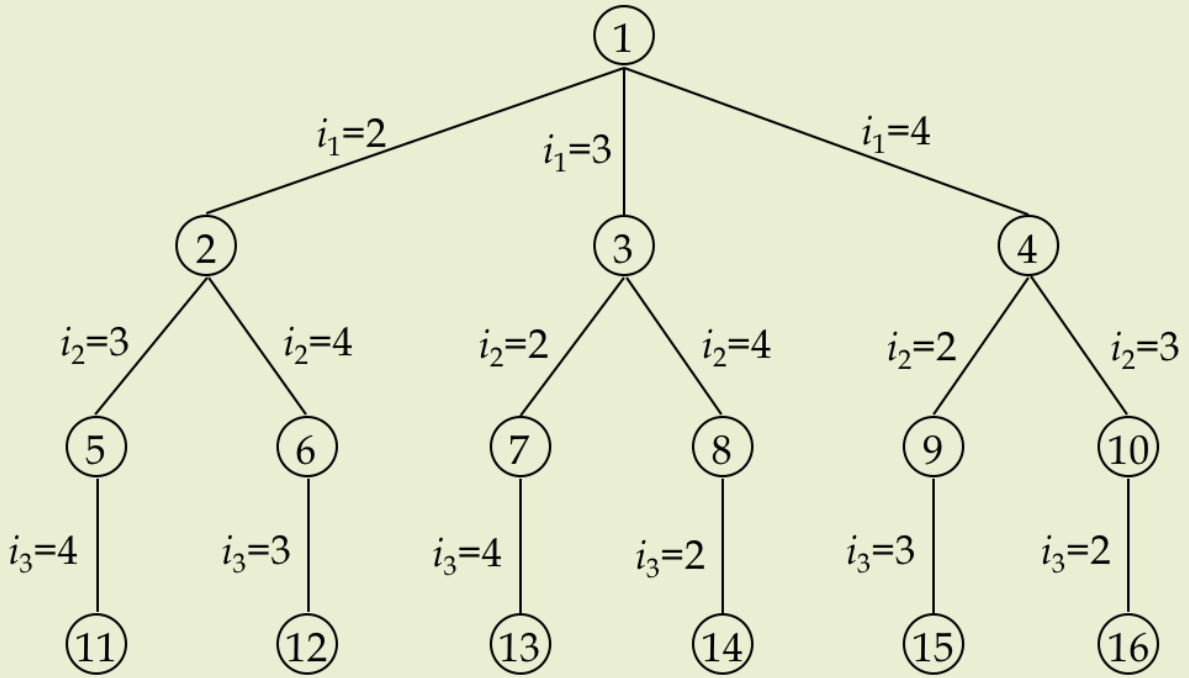
- Espacio de soluciones: $E = \{1, \pi, 1 \mid \pi$ es una permutación de $(2, 3, \ldots, n)\}$
- Tamaño: $|E| = (n-1)!$

Reducción del espacio de soluciones

$E = \{1, \pi, 1 \mid \pi = i_1, i_2, \ldots, i_{n-1},$ es una permutación de $(2, 3, \ldots, n)$ tal que $(i_j, i_{j+1}) \in A, 0 \leq j \leq n-1$ y $i_0 = i_n = 1\}$.

Representación del espacio de estados

Caso de un grafo completo $|V| = 4$



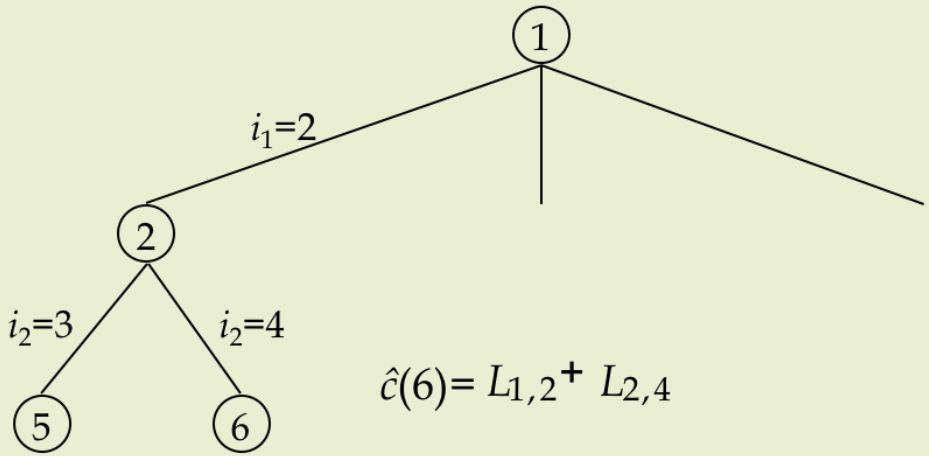
Cada hoja es una solución y representa el viaje definido por el camino desde la raíz hasta la hoja.

Definición de la función de coste

- La función c debe ser tal que el nodo solución x con valor mínimo de $c(x)$ corresponda a un recorrido de longitud mínima
- Por ejemplo:
  - Si x es una hoja, entonces $c(x)$ es la longitud del hamiltoniano definido por el camino desde la raíz hasta x
  - Si x no es una hoja, entonces $c(x)$ es el coste de una hoja de coste mínimo del subárbol con raíz x

Definición de la función de estimación

- La función $\hat{c}$ debe ser tal que $\hat{c}(x) \leq c(x)$, para cada x
- Una función muy fácil:
  - $\hat{c}(x)$ es la distancia total del recorrido definido por el camino desde la raíz hasta x



- Si es una hoja, es obvio que $\hat{c}(x) = c(x)$

Definición de la función de estimación

Mejora de $\hat{c}(x)$

Matriz de distancias reducidas

- Una fila (columna) de la matriz de distancias se dice **reducida** si sus elementos son no negativos y contiene al menos un 0 (cero)
- Una matriz de distancias se dice reducida si cada fila y columna es reducida

- Para cada $k, 1 \leq k \leq n$, todo circuito hamiltoniano incluye exactamente un arco de la forma $(k, \cdot)$ y exactamente un arco de la forma $(\cdot, k)$
- Si se resta una constante t de cada elemento de una fila (columna) de la matriz de distancia, la longitud de todo hamiltoniano se reduce exactamente en t y un hamiltoniano de distancia mínima lo sigue siendo

Definición de la función de estimación

Ejemplo de reducción

- Reducción fila 1, $t = 10$

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

- Se repite el proceso hasta obtener la matriz de distancia reducida
- Reducción: $L = 25$

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

La cantidad L total restada es una cota inferior de la longitud de un hamiltoniano de longitud mínima, luego sirve como estimación de $\hat{c}(x)$ para el nodo x raíz.

Definición de la función de estimación

Cálculo de $\hat{c}(x)$ para los nodos x intermedios (no raíz y no hoja)

- Sea A la matriz de distancia reducida para el nodo y
- Sea x un hijo de y que corresponda a incluir el arco (i, j) en el recorrido y que no sea hoja
- La matriz B reducida para x se calcula de la siguiente forma:

1. Cambiar todos los elementos de la fila i y de la columna j de A por ∞
2. Cambiar el elemento $A(j, 1)$ por ∞ para evitar considerar el arco $(j, 1)$
3. B es la matriz que se obtiene al reducir todas las filas y columnas de la matriz resultante (excepto aquéllas formadas sólo por ∞).
   Si r es el valor total restado:

$$\hat{c}(x) = \hat{c}(y) + A(i, j) + r$$

Definición de la función de poda

- Una función de poda trivial:
  - Inicialmente, $\mathcal{U} = \infty$
  - Se modifica $\mathcal{U}$ sólo cuando se llega a un nodo x que es hoja (solución factible), entonces:

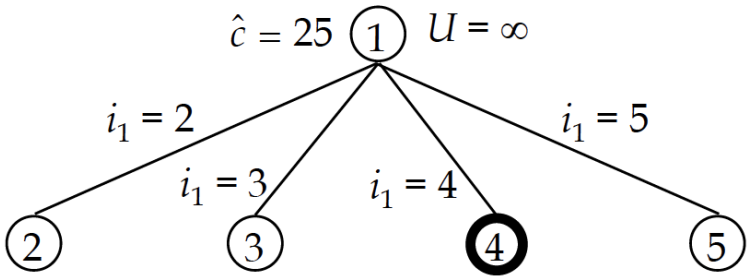
$$U = min\{\mathcal{U}, c(x)\}$$

- Se puede pensar en mejores funciones de poda

Ejemplo

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$ Grafo original $$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$ Reducción: $L = 25$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$



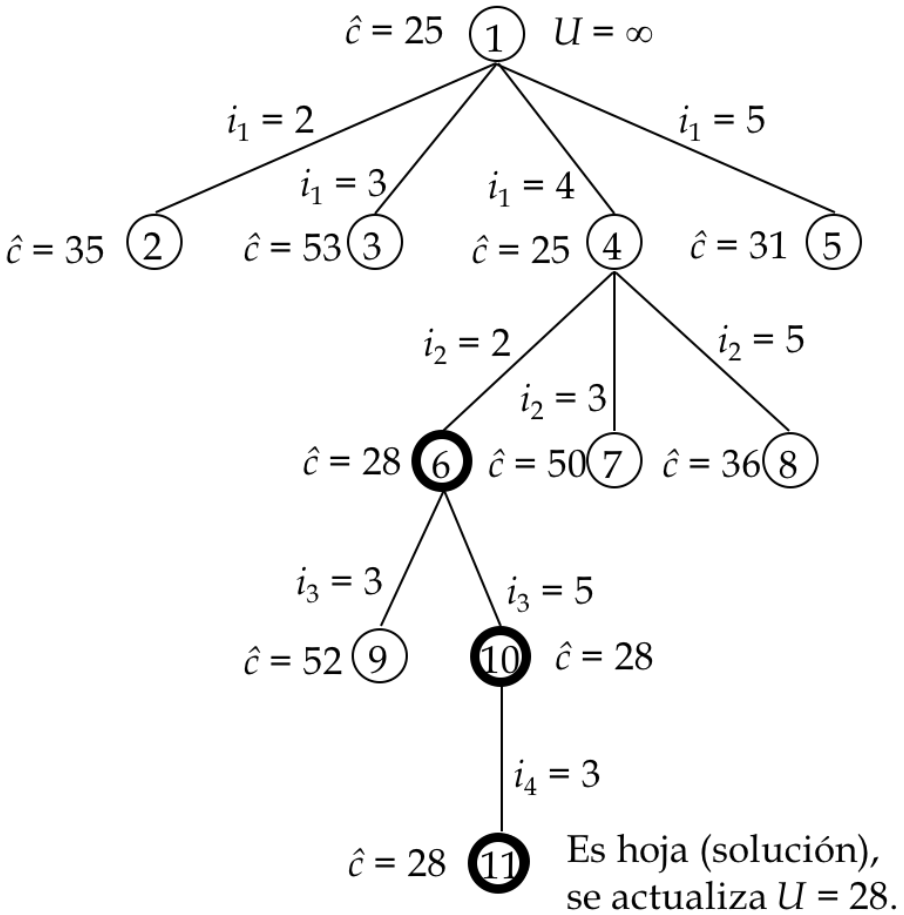
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

$\hat{c} = 25+10 = 35$ $\hat{c} = 25+17+11 = 53$ $\hat{c} = 25$ $\hat{c} = 25+1+5 = 31$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

Ejemplo



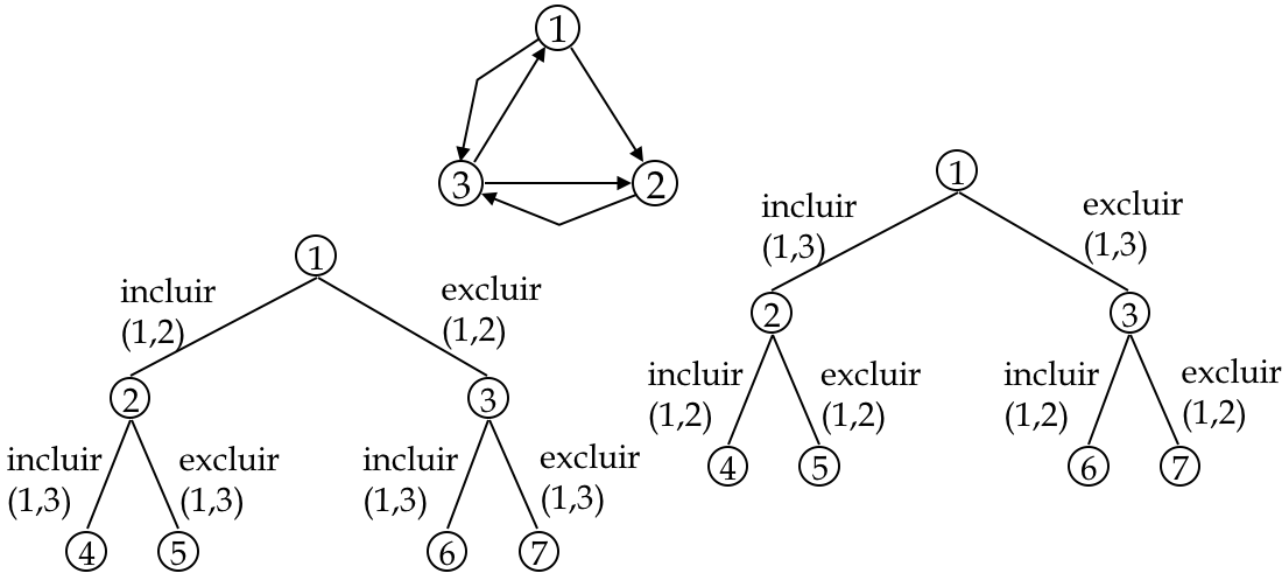
- El siguiente nodo a expander sería (5) pero $\hat{c}(5) > \mathcal{U}$ luego el algoritmo termina
- El hamiltoniano mínimo es 1,4,2,5,3,1

Otras soluciones

- Un hamiltoniano es un **conjunto** de n arcos
- Además, para cada vértice $i, 1 \leq i \leq n$, debe haber en ese conjunto exactamente un arco de la forma $(i, \cdot)$ y uno de la forma $(\cdot, i)$
- Representación del espacio de estados como un árbol binario
  - Un hijo izquierdo representa la **inclusión** de un determinado arco en el hamiltoniano
  - Un hijo derecho representa la **exclusión** de ese arco

Representaciones con árboles binarios

- Hay distinto árboles dependiendo de qué arco se elige para incluir/excluir en cada paso



- ¿Cómo elegir el arco por el que empezar?
  - Depende de los datos del problema
- Árbol de estados **dinámico**: el método de construcción depende de los datos del problema

Representaciones con árboles binarios

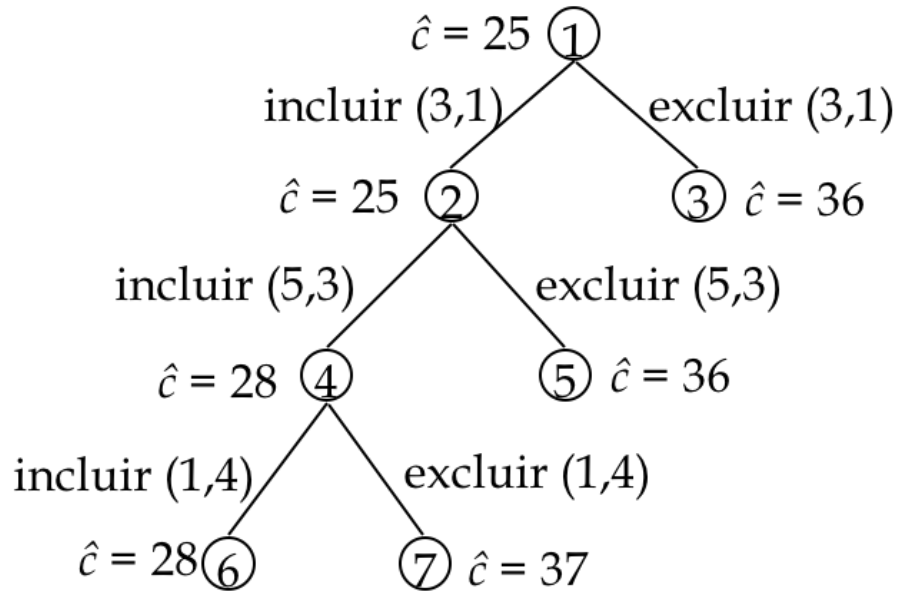
- Si se elige, para empezar, el arco (i, j)
  - El subárbol izquierdo representa todos los recorridos que incluyen (i, j)
  - El subárbol derecho representa todos los recorridos que no lo incluyen
  - Si hay un recorrido óptimo incluido en el subárbol izquierdo, entonces sólo faltan por seleccionar $n - 1$ arcos para encontrarlo
  - Mientras que si todos los recorridos óptimos están en el derecho, hay que seleccionar todavía n arcos
- Hay que intentar seleccionar un arco que tenga **la máxima probabilidad de estar en el recorrido óptimo**
- Varias heurísticas posibles:
  - Elegir un arco que produzca un subárbol derecho cuya raíz tenga $\hat{c}(x)$ máxima
  - Elegir un arco tal que la diferencia entre $\hat{c}(x_{izquierdo})$ y $\hat{c}(x_{derecho})$ sea máxima

Representaciones con árboles binarios

Heurística del máximo coste estimado

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

Grafo original



- Al nodo (6) se han elegido ya tres arcos: $(3,1)$, $(5,3)$ y $(1,4)$
- Para los restantes, sólo queda ya una opción: $(4,2)$ y $(2,5)$
- Hamiltoniano: 5,3,1,4,2,5 con distancia total: 28 (así, $\mathcal{U} = 28$)
- El siguiente nodo a expander es (3) con $\hat{c}(3) > \mathcal{U}$. El algoritmo acaba.

Observaciones

- En el último ejemplo, el método de ramificación y poda se usa sólo para subárboles grandes
- Una vez que se llega a subárboles pequeños (4 ó 6 nodos) es más eficientes construirlos enteros y calcular el valor exacto de la función de coste

No hay forma de calcular analíticamente cuál de las soluciones del problema del viajante de comercio es más eficiente.

Consideraciones finales sobre eficiencia

- ¿Es posible disminuir en algún caso el número de nodos generados mediante la expansión de nodos x con $\hat{c}(x) > \mathcal{U}$?

Nunca porque el valor de $\mathcal{U}$ no será modificado si se expande tal nodo x, y es el valor de $\mathcal{U}$ que determina qué nodos se expanden en el futuro.

- Si hay dos funciones de poda $\mathcal{U}_1$, $\mathcal{U}_2$ diferentes y $\mathcal{U}_1 < \mathcal{U}_2$ entonces **siempre** es mejor usar $\mathcal{U}_1$ (nunca dará lugar a más nodos)
- ¿Si se usa una función de estimación mejor, decrece el número de nodos a expandir? ($\hat{c}_2$ es mejor que $\hat{c}_1$ si $\hat{c}_1 \leq \hat{c}_2 \leq c$)

No necesariamente. Si se usa una función de estimación mejor con la estrategia de ramificación de **mínimo coste**, el número de nodos generados puede crecer.