



UNIVERSIDAD
DE GRANADA



Algorítmica

Grado en Ingeniería Informática

Tema 5 – Algoritmos para exploración en grafos

Manuel Pegalajar Cuéllar

manupc@ugr.es

Departamento de Ciencias de la
Computación e Inteligencia Artificial

<http://decsai.ugr.es>

*Este documento está protegido por la
Ley de Propiedad Intelectual (
Real Decreto Ley 1/1996 de 12 de abril).
Queda expresamente prohibido su uso o
distribución sin autorización del autor.*

Objetivos del tema

- Plantearse la búsqueda de varias soluciones distintas para un mismo problema y evaluar la bondad de cada una de ellas.
- Saber ver al árbol de estados como una representación lógica del conjunto de todas las posibles soluciones de un problema.
- Conocer las técnicas de exploración de grafos (vuelta atrás y ramificación y poda) y su aplicación en la resolución de problemas, entendiendo sus características principales y las diferencias entre ellas.
- Comprender y saber aplicar el uso de cotas para reducir el espacio de búsqueda en las técnicas de exploración en grafos.
- Conocer los criterios de aplicación de cada una de las distintas técnicas de diseño de algoritmos.

Estudia este tema en...

- G. Brassard, P. Bratley, “Fundamentos de Algoritmia”, Prentice Hall, 1997, pp. 319-363
- J.L. Verdegay: Lecciones de Algorítmica. Editorial Técnica AVICAM (2017).

Anotación sobre estas diapositivas:

El contenido de estas diapositivas es esquemático y representa un apoyo para las clases presenciales teóricas. No se considera un sustituto para apuntes de la asignatura.

Se recomienda al alumno completar estas diapositivas con notas/apuntes propios, tomados en clase y/o desde la bibliografía principal de la asignatura.



UNIVERSIDAD
DE GRANADA

Algoritmos para exploración en grafos



1. Introducción
2. Representación de espacios de estados
3. La técnica Backtracking
4. Las 8 reinas
5. Resolución de Sudokus
6. El viajante de comercio
7. La técnica de Branch&Bound
8. Asignación de tareas
9. El viajante de comercio



DECSAI

- Para una gran cantidad de problemas, no existe un algoritmo conocido que los resuelva de forma eficiente.
- En estos casos, la resolución del problema pasa por una exploración directa de todas (o gran parte de) las posibilidades de llegar a una solución.
- Muchos de estos problemas se pueden representar como un grafo: Por eso es necesario conocer las técnicas para su exploración.



- Los recorridos sobre un grafo ya se conocen: Se han estudiado en la asignatura de Estructuras de Datos. Ejemplo de **recorrido en profundidad**:

Procedimiento RecorridoProfundidad($G=(V,A)$)

Para cada v en V **hacer** $\text{marca}[v] = \text{NoVisitado}$

Para cada v en V **hacer**:

Si $\text{marca}[v] = \text{NoVisitado}$, **entonces** $\text{rp}(v)$

Fin-Para

Procedimiento $\text{rp}(\text{vértice } v)$

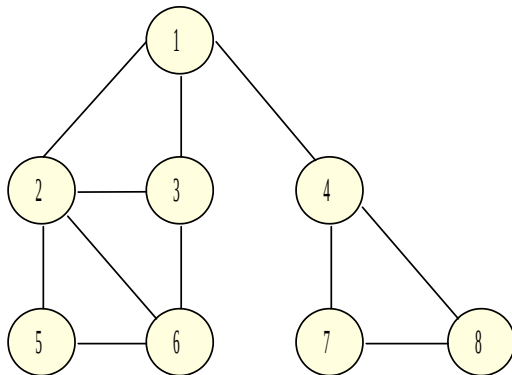
$\text{Marca}[v] = \text{visitado}$

Para cada w adyacente a v **hacer**:

Si $\text{marca}[w] = \text{NoVisitado}$, **entonces** $\text{rp}(w)$

Fin-Para

■ Ejemplo de ejecución: **recorrido en profundidad**:



■ Ejecución: (comienzo desde el nodo 1)

■ rp(1) Se visita el nodo 1 y pasa al adyacente

■ rp(2) Llamada recursiva, visita 3

■ rp(3) Llamada recursiva, visita 6

■ rp(6) Llamada recursiva, visita 5

■ rp(5) Sin nodos adyacentes

■ rp(4) Llamada recursiva de rp(1), visita 7

■ rp(7) Llamada recursiva, visita 8

■ rp(8) sin nodos que visitar

■ Ejemplo de recorrido en anchura:

Procedimiento RecorridoAnchura($G=(V,A)$)

Para cada v en V **hacer** $\text{marca}[v] = \text{NoVisitado}$

Para cada v en V **hacer:**

Si $\text{marca}[v] = \text{NoVisitado}$, **entonces** $\text{ra}(v)$

Fin-Para

Procedimiento $\text{ra}(v)$

$Q = \text{Cola vacía}$

$\text{marca}[v] = \text{Visitado};$

Insertar v en Q

Mientras Q no esté vacía, **hacer:**

$u = \text{Quitar primer elemento de } Q$

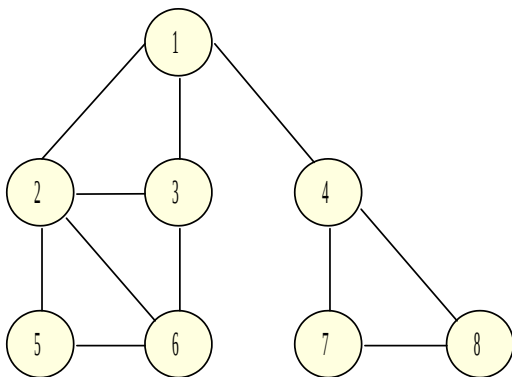
Para cada nodo w adyacente a u no visitado, **hacer:**

$\text{marca}[w] = \text{Visitado};$ Insertar w al final de Q

Fin-Para

Fin-Mientras

■ Ejemplo de ejecución: **recorrido en anchura:**



Paso	V visitado	Q
1	1	2, 3, 4
2	2	3, 4, 5, 6
3	3	4, 5, 6, 7, 8
4	4	5, 6, 7, 8
5	5	6, 7, 8
6	6	7, 8
7	7	8
8	8	

- Las técnicas de BackTracking, Branch&Bound (ramificación y poda) y de árboles para juegos hacen una exploración de todos los posibles estados de un problema hasta llegar a la solución.
- Pueden implementar un recorrido en profundidad, en anchura o variantes que ayuden en la búsqueda

Ventajas

- Fáciles de diseñar.
- Fáciles de implementar.
- Resuelven una gran cantidad de problemas.

Inconvenientes

- Ineficientes en tiempo y en espacio.



UNIVERSIDAD
DE GRANADA

Algoritmos para exploración en grafos

1. Introducción
- » 2. Representación de espacios de estados
3. La técnica Backtracking
4. Las 8 reinas
5. Resolución de Sudokus
6. El viajante de comercio
7. La técnica de Branch&Bound
8. Asignación de tareas
9. El viajante de comercio



DECSAI

- Usaremos árboles/grafos para representar posibles estados de un problema y los movimientos/acciones a realizar para pasar de un estado a otro.

Representación en espacios de estados

- Cada nodo del grafo representa un estado del problema.
- Contiene toda la información necesaria para poder diferenciar el estado del problema de cualquier otro.
- Las aristas se asocian a acciones o decisiones a tomar.
- Una acción (arista), aplicada sobre un estado del problema da como resultado un cambio de estado a otro conocido.

Ejemplo: El juego de los palillos

Inicialmente, hay n palillos sobre la mesa, y dos jugadores A y B. El jugador A comienza el juego quitando 1, 2 ó 3 palillos. Le sigue el jugador B, que también podrá quitar 1, 2 ó 3 palillos. El turno vuelve al jugador A, y estas acciones se repiten hasta que quede un único palillo en la mesa. Aquel que quite este último palillo pierde el juego.

Ejemplo: El juego de los palillos. Representación


- Representaremos el juego como un grafo,
- Cada nodo representa un **estado** del problema (el número de palillos sobre la mesa).
- Cada arista representa un **movimiento** (quitar 1, 2 ó 3 palillos).
- La aplicación de un movimiento conlleva un cambio de estado entre uno inicial y otro final.
- (Concretamente, para esta versión del juego se puede dar una estrategia óptima para el jugador 1 haciendo un estudio algebraico. Pero supongamos que no, sólo porque el ejemplo resulta ilustrativo sobre qué es un árbol de estados...)




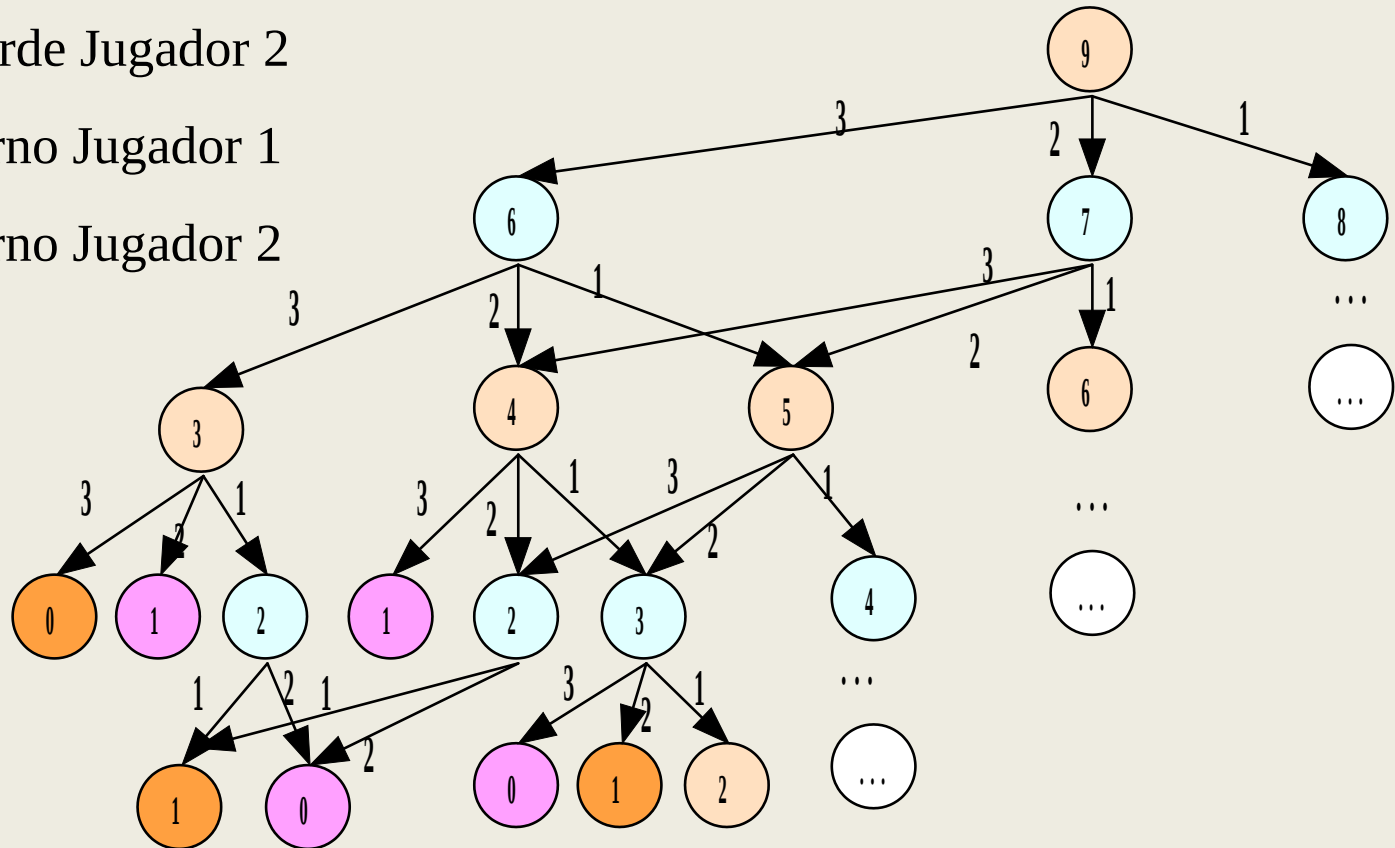
Ejemplo: El juego de los palillos. Representación con 9 palillos

 Pierde Jugador 1

 Pierde Jugador 2

 Turno Jugador 1

 Turno Jugador 2





UNIVERSIDAD
DE GRANADA

Algoritmos para exploración en grafos

1. Introducción
2. Representación de espacios de estados
- » 3. La técnica Backtracking
4. Las 8 reinas
5. Resolución de Sudokus
6. El viajante de comercio
7. La técnica de Branch&Bound
8. Asignación de tareas
9. El viajante de comercio



DECSAI

Idea general

- Hacer una búsqueda exhaustiva sobre grafos (árboles) **dirigidos y acíclicos**, mediante recorrido en profundidad.
- Se realiza una poda de ramas poco prometedoras en el grafo para acelerar la búsqueda.
- Las soluciones se expresan dependiendo de la representación del problema, como vectores $T=(x_1, x_2, x_3, \dots, x_t)$. Representan a la secuencia de decisiones (arcos del grafo) tomadas desde el estado inicial hasta el estado final objetivo.
- **Criterios de parada:** Depende del objetivo perseguido en el problema: a) **encontrar todas las soluciones al problema**, o b) **Encontrar alguna solución**.



Componentes de diseño Backtracking

- Buscar una **representación** del tipo $T=(x_1, x_2, \dots, x_t)$, para las soluciones del problema.
- Diseñar las **restricciones implícitas**: Son los valores que cada valor x_i puede tener para construir la solución.
- Identificar las **restricciones explícitas**: Restricciones externas al proceso de encontrar la solución.
- Diseñar la **estructura del árbol/grafó implícito** que define los estados y transiciones entre estados de búsqueda de soluciones.
- Diseñar la **función objetivo**: Criterio de parada para encontrar la solución/soluciones requerida/s.
- Diseñar una **función de poda** $B_k(x_1, x_2, \dots, x_k)$ para eliminar la exploración de ramas que deriven en soluciones inadecuadas.

Plantilla general de un algoritmo backtracking

Procedimiento BackTracking($k, T[x_1..x_t]$)

Para cada valor v posible de la variable x_k **hacer:**

Si es factible $T \cup \{v\}$ **entonces**

Si $T \cup \{v\}$ es solución **entonces Devolver** $T \cup \{v\}$

en otro caso,

Si $k < t$ **entonces**

$u = \text{BackTracking}(k+1, T \cup \{v\})$

Si u es solución **entonces Devolver** u

Fin-Si

Fin-En otro caso

Fin-Si

Fin-Para

Devolver No hay solución



UNIVERSIDAD
DE GRANADA

Algoritmos para exploración en grafos

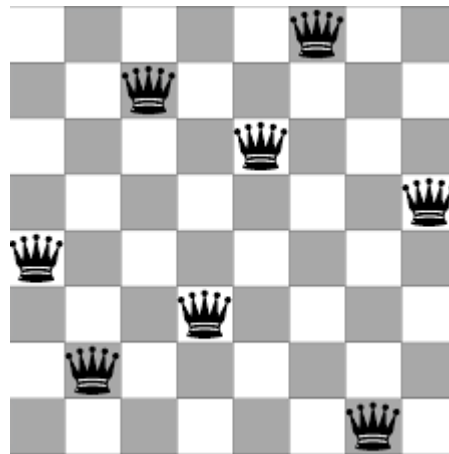
1. Introducción
2. Representación de espacios de estados
3. La técnica Backtracking
- » 4. Las 8 reinas
5. Resolución de Sudokus
6. El viajante de comercio
7. La técnica de Branch&Bound
8. Asignación de tareas
9. El viajante de comercio



DECSAI

Enunciado

En un tablero de ajedrez de tamaño $N \times N$, se desea colocar N reinas sin que ningún par se dé jaque entre sí.



Las 8 reinas: Componentes Backtracking (I)

Representación: $T(x_1, x_2, \dots, x_N)$ es un vector donde cada componente i representa una columna del tablero y cada valor x_i es la fila donde se colocará la reina de la i -ésima columna.

Restricciones implícitas: x_i tendrá valores entre 1 y N (inclusive), la fila donde se colocará la i -ésima reina.

Restricciones explícitas: No puede haber 2 reinas en la misma fila, en la misma columna, o en la misma diagonal.



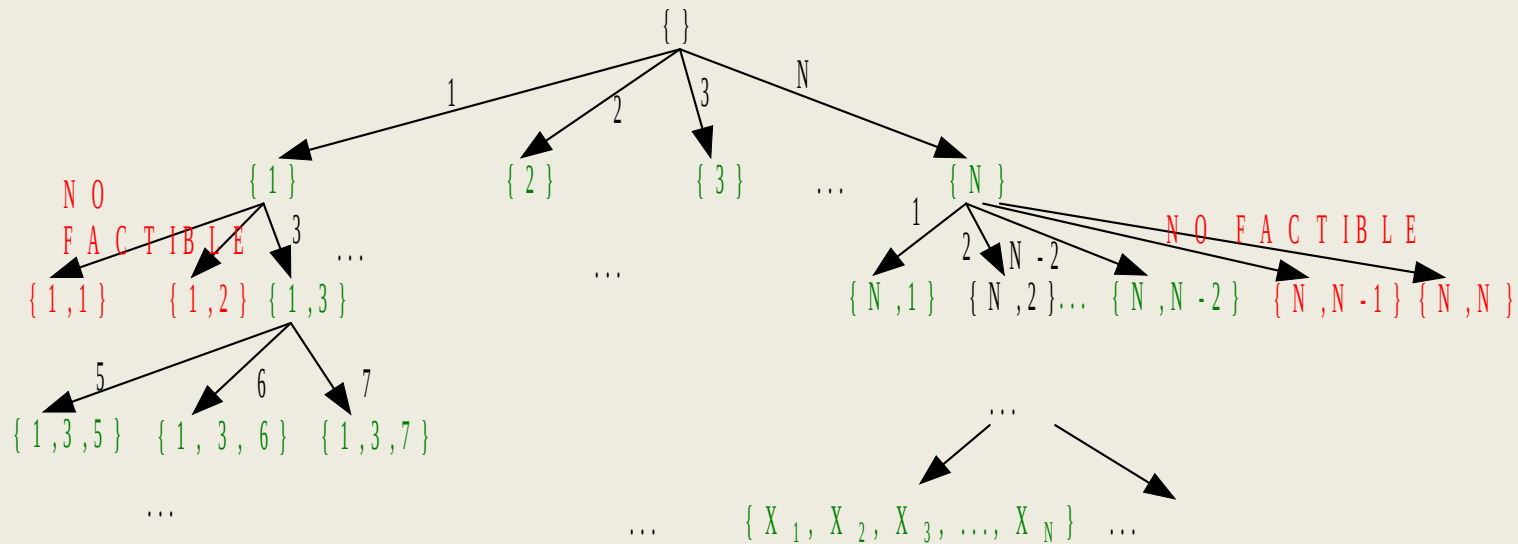
Las 8 reinas: Componentes Backtracking (II)

Representación del árbol implícito:

- El **estado inicial** será el vector $T=(0)_N$, donde el 0 indica que no hay cada nivel i del árbol asignamos la reina de la i -ésima columna a una fila del tablero.
- En **cada nivel** del árbol trataremos de solucionar la posición i del vector (i -ésima columna).
- Cada estado tiene las siguientes **acciones (arcos)**: Poner la reina de la columna i asociada al estado (nivel del estado en el árbol) en cada posición de 1 a N .
- El **estado final** o **solución** se dará cuando todo el vector esté relleno cumpliendo con restricciones implícitas y explícitas.

Las 8 reinas: Componentes Backtracking (III)

Representación del árbol implícito:



Las 8 reinas: Componentes Backtracking (IV)

- **Función objetivo:** Encontrar un vector $T(x_1, x_2, \dots, x_N)$ que sea solución al problema (N reinas colocadas en el tablero sin darse jaque).
- **Función de poda:** Al hacer $T(x_1, x_2, \dots, x_{k-1}) \cup x_k$, x_k debe cumplir:
 - Implícitamente por la representación usada para T, no puede haber 2 reinas en la misma columna.
 - No existe x_i , con $i < k$, tal que $x_i = x_k$. (Dos reinas en la misma fila).
 - No existe x_i , con $i < k$, tal que $\text{abs}(x_i - x_k) = \text{abs}(i - k)$.



Las 8 reinas: Diseño del algoritmo

Procedimiento Soluc(N)

$T = (0, 0, 0, \dots, 0)_N$

Devolver NReinas(1, T)

Procedimiento BTNReinas(k, T[1..N])

Para $i=1..N$, **hacer**:

$T[k] = i$

Si T es factible **entonces**

Si $k=N$ **entonces Devolver** T

en otro caso,

Si $k < N$ **entonces**

$T = \text{BTNReinas}(k+1, T)$

Si T es solución **entonces Devolver** T

Fin-Si

Fin-En otro caso

Fin-Si

Fin-Para

Devolver No hay solución

Las 8 reinas: Implementación.

```

19 bool SolucionarNReinas(int *sol, int k, int N) {
20
21     if (k>=N) return true; // Todo relleno. Solución encontrada
22
23     // Probamos con cada posición (fila) para la columna k
24     for (int i= 0; i<N; i++) {
25
26         sol[k]= i;
27
28         // Comprobamos factibilidad
29         if (factible(sol, k)) {
30             bool solucion= SolucionarNReinas(sol, k+1, N);
31             if (solucion) // Si se ha encontrado solución, salimos y no continuamos el bucle
32                 return true;
33         }
34     }
35
36     // No se ha encontrado solución: Vuelta atrás
37     sol[k]= -1;
38     return false;
39 }

```



UNIVERSIDAD
DE GRANADA

Algoritmos para exploración en grafos

1. Introducción
2. Representación de espacios de estados
3. La técnica Backtracking
4. Las 8 reinas
- » 5. Resolución de Sudokus
6. El viajante de comercio
7. La técnica de Branch&Bound
8. Asignación de tareas
9. El viajante de comercio



DECSAI

Enunciado (I)

Un Sudoku es un pasatiempo matemático que consiste en rellenar con números del 1 al 9 una tabla de 9x9 elementos, dividida en subcuadrículas de 3x3, con casillas ya rellenadas previamente. Las normas del pasatiempo son:

- ❖ No puede haber dos casillas en la misma fila con el mismo número.
- ❖ No puede haber dos casillas en la misma columna con el mismo número.
- ❖ No puede haber dos casillas, en la misma subcuadrícula de 3x3, con el mismo número.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



Enunciado (II)

Para solucionar un sudoku, es necesario escribir números entre 1 y 9 en todas las casillas que quedan vacías, de modo que cumplan las 3 reglas anteriores.

Las casillas que pueden modificarse son sólo aquellas que al comienzo están vacías.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



Resolución de Sudokus: Componentes Backtracking (I)

Representación: $T(x_1, x_2, \dots, x_N)$, con $N=9*9=81$, se representará como una matriz $S(i, j)$ conteniendo el dígito entre 1 y 9 que haya en la fila i , columna j del tablero, con $1 \leq i, j \leq 9$.

- La componente del vector x_k se asociará a la fila de la matriz S con la siguiente proyección: $k=(i*(9-1)+j)$
- Para diferenciar entre casillas modificables y no modificables, las que no se puedan alterar tendrán el valor en negativo.
- Las casillas que no tengan valor asignado tendrán valor 0

Restricciones implícitas: x_k tendrá valores entre 1 y 9 (inclusive), el dígito asociado a la k -ésima celda de la matriz $S(i, j)$



Resolución de Sudokus: Componentes Backtracking (II)

Restricciones explícitas:

- No puede haber 2 valores distintos de 0 iguales (en valor absoluto), en la misma fila i ; es decir: $S(i,j) \neq S(i,k)$ para todo i .
- No puede haber 2 valores distintos de 0 iguales (en valor absoluto), en la misma columna i ; es decir: $S(j, i) \neq S(k, i)$ para todo i .
- No puede haber 2 valores distintos de 0 iguales (en valor absoluto), en la misma subcuadrícula de 3×3 . Es decir:

$S(i,j) \neq S(a,b)$ para todo $S(i,j), S(a,b)$ distintos de 0 tal que:

- $(i,j), (a,b)$ estén en $(1..3, 1..3)$ o
- $(i,j), (a,b)$ estén en $(1..3, 4..6)$ o
- $(i,j), (a,b)$ estén en $(1..3, 7..9)$ o...
- ... $(i,j), (a,b)$ estén en $(7..9, 7..9)$

Resolución de Sudokus: Componentes Backtracking (II)

■ Representación del árbol implícito:

- El **estado inicial** será la matriz $S(i,j)=0$ en todas las casillas vacías, salvo en las que no se pueden modificar que tendrán valores desde -1 hasta -9.
- En **cada nivel** del árbol trataremos de solucionar la posición **k** del vector, con $k=(i*(9-1)+j)$, recorriendo la matriz de izquierda a derecha y de arriba abajo.
- Cada estado tiene las siguientes **acciones (arcos)**: Si la casilla está vacía, asignar un número entre 1 y 9 válido.
- El **estado final** o **solución** se dará cuando toda la matriz S esté rellena cumpliendo con restricciones implícitas y explícitas.

Resolución de Sudokus: Componentes Backtracking (IV)

■ **Función objetivo:** Encontrar una matriz $S(1..9,1..9)$ que sea solución al problema (todas las casillas del Sudoku rellenas).

■ **Función de poda:** Al hacer $S(i,j) = \text{valor}$, se deberá podar si:

■ Dos casillas de la misma fila no vacías tienen igual valor. Es decir, si $S(i,j) = S(i,k)$ para todo i , y $S(i,j) \neq 0$

■ Dos casillas de la misma columna no vacías tienen igual valor. Es decir, si $S(j,i) = S(k,i)$ para todo i , y $S(j,i) \neq 0$

■ Dos casillas del mismo subrecuadro de 3×3 no vacías tienen el mismo valor. Es decir, $S(i,j) = S(a,b)$, $S(i,j) \neq 0$, y:

■ $\text{Floor}(i/3) = \text{Floor}(a/3)$ y $\text{Floor}(j/3) = \text{Floor}(b/3)$,

Donde $\text{Floor}(x)$ representa parte entera.

Resolución de sudokus: Diseño del algoritmo

Procedimiento Soluc(N)

S= Matriz inicial

Devolver Sudoku(1, S)

Procedimiento Sudoku(k, S[1..9, 1..9])

Calcular $j = (k-1)\%9+1$, $i = (k-1)/9+1$

Si $(k > 81)$ **devolver** S; **en otro caso:**

Si $(k \leq 0)$ **devolver** Sudoku(k+1, S); **en otro caso:**

Para $v = 1..9$, **hacer:**

$S(i, j) = v$

Si S es factible **entonces**

$S' = \text{Sudoku}(k+1, S)$

Si S' es solución **Devolver** S'

Fin-Si

Fin-Para

$S(i, j) = 0$

Fin-En otro caso

Devolver No hay solución

Resolución de sudokus: Implementación.

```

63  bool Sudoku(int k, int T[9][9]) {
64
65      if (k>80)
66          return true; // En T está la solución
67
68      int i, j; // Coordenadas de la matriz correspondientes a k
69
70      do {
71          j= k%9;
72          i= k/9;
73          if (T[i][j] < 0)
74              k++;
75          if (k>80)
76              return true;
77      } while (T[i][j]<0 );
78
79      // Para cada valor posible en el nivel k...
80      for (int v= 1; v<=9; v++) {
81
82          T[i][j]= v;
83
84          // Si es factible, llamar al nivel siguiente
85          if (esfactible(T, i, j) && Sudoku(k+1, T))
86              return true;
87      }
88
89      T[i][j]= 0;
90      return false;
91  }

```



UNIVERSIDAD
DE GRANADA

Algoritmos para exploración en grafos

1. Introducción
2. Representación de espacios de estados
3. La técnica Backtracking
4. Las 8 reinas
5. Resolución de Sudokus
- » 6. El viajante de comercio
7. La técnica de Branch&Bound
8. Asignación de tareas
9. El viajante de comercio



DECSAI

Enunciado del problema

Sea un grafo $G=(V,A)$, **no dirigido**, **completo**, con V vértices y A aristas, donde las **aristas están ponderadas con pesos no negativos**.

El problema del viajante de comercio consiste en:

“Encontrar el circuito hamiltoniano minimal del grafo G .”

Ejemplo de problema tipo del viajante de comercio

Un repartidor de periódicos sale todas las mañanas de su almacén, y debe repartir la mercancía por todos los kioscos de la ciudad. Diseñar una ruta que le permita repartir los periódicos en todos los kioscos y terminar en el almacén de donde partió, sin repetir ningún kiosco y haciendo que la ruta completa tenga un coste mínimo.

El viajante de comercio: Componentes Backtracking (I)

■ **Representación:** $T(x_1, x_2, \dots, x_N)$, con N =número de nodos del grafo.

■ La componente del vector x_k se asociará al nodo que hay que visitar en la k -ésima posición.

■ **Restricciones implícitas:** x_k tendrá valores entre 1 y N (inclusive)

■ **Restricciones explícitas:**

■ No puede visitarse el mismo nodo dos veces. Formalmente:

$$x_i \neq x_j \text{ para todo } i, j$$

■ **Función objetivo:** Encontrar un vector $T(x_1, x_2, \dots, x_N)$ tal que el circuito hamiltoniano que representa sobre el grafo sea mínimo.

■ **Función de poda:** Al hacer $T(x_1, x_2, \dots, x_{k-1}) \cup x_k$, x_k debe cumplir: $x_k \neq x_i$ para todo $i < k$

El viajante de comercio: Componentes Backtracking (II)

Representación del árbol implícito:

- El **estado inicial** será el vector $T = \{1, 0, 0 \dots 0\}_N$ en todas las casillas vacías, salvo en la primera, que supondrá que se parte desde el nodo 1.
- En **cada nivel** del árbol trataremos de solucionar la posición **k** del vector, asignando un nodo no visitado.
- Cada estado tiene las siguientes **acciones (arcos)**: Tantos arcos como nodos aún queden por visitar y que puedan ser sucesores del nodo del estado actual.
- El **estado final** o **solución** se dará cuando el vector esté relleno y el circuito hamiltoniano que representa tenga longitud mínima.

Resolución de sudokus: Diseño del algoritmo

Procedimiento [MejorSol]= Soluc($G=<V,A>$)

$N = |V|;$

$T = \{1, 0, 0, \dots, 0\}_N$

MejorSolucion = $\{0, 0, 0, \dots, 0\}_N$, con $\text{coste}(\text{MejorSolucion}) = \infty$

TSP($G, \text{MejorSol}, T, 2$)

Devolver MejorSol]

Procedimiento TSP($G = <V,A>, MS, T, k$)

Soluciona la componente k del vector T sobre el grafo G .

El parámetro MS representa a la mejor solución encontrada hasta el momento.

Resolución de sudokus: Diseño del algoritmo

Procedimiento TSP($G = \langle V, A \rangle$, MS, T, k)

Para cada nodo $i=1..|V|$

Si i no ha sido utilizado en $T[1..k-1]$, **entonces:**

$T[k] = i$;

Si $k=|V|$, **entonces:**

Si $\text{coste}(T)$ mejor que $\text{coste}(MS)$, **entonces:**

Actualizar $MS = T$

Fin-Si

En otro caso, si $k < |V|$:

Resolver TSP(G , MS, T, $k+1$)

Fin-En otro caso

Fin-Si

Fin-Para

Devolver MS



UNIVERSIDAD
DE GRANADA

Algoritmos para exploración en grafos

1. Introducción
2. Representación de espacios de estados
3. La técnica Backtracking
4. Las 8 reinas
5. Resolución de Sudokus
6. El viajante de comercio
- » 7. La técnica de Branch&Bound
8. Asignación de tareas
9. El viajante de comercio



DECSAI

Terminología

- **Nodo vivo:** Nodo del espacio de soluciones del que no se han generado/visitado aún todos sus hijos.
- **Nodo en curso (o en expansión):** nodo del que se están generando (visitando) hijos.
- **Nodo muerto:** nodo del que no se van a generar más hijos porque:
 - A) es un nodo hoja,
 - B) se poda,
 - C) no producirá una solución mejor que la solución en curso



Diferencias entre Branch&Bound y Backtracking

- **Backtracking:** Tan pronto se puede pasar a generar/visitar un nodo, se hace. *(Los únicos nodos vivos son lo que hay desde la raíz del árbol hasta el nodo actual)*
- **Branch&Bound:** Se generan todos los nodos hijos del nodo en curso (nodo actual) antes de que cualquier otro nodo vivo pase a ser el nodo en curso. Por tanto, se deben almacenar los nodos vivos en una estructura de datos auxiliar: **Lista de nodos vivos.**

UNIVERSITAT DE VALÈNCIA



PLUS

ULTRA

Diferencias entre Branch&Bound y Backtracking

■ **Ventajas de Backtracking:**

- Es más fácil de implementar.
- Tiene menos requisitos de memoria.

■ **Ventajas de Branch&Bound:**

- Revisa menos estados para llegar a la solución.
- Es más eficiente

E
>
-
Z



PLUS

ULTRA

Componentes de diseño Branch&Bound

- **Representación** de una solución en un vector (x_1, x_2, \dots, x_n)
- **Función objetivo** que determina si la solución actual es óptima.
- **Restricciones implícitas:** Valores de cada x_i
- **Restricciones explícitas:** Las que no dependen de la representación del problema.
- **Función de elección** para seleccionar qué nodo es mejor para considerarlo en curso.
- **Cálculo de cotas**, para eliminar partes del árbol que no van a proporcionar la solución o para seleccionar el camino más prometedor.
- Organizar el problema en un **árbol de búsqueda**.

Cálculo de cotas en Branch&Bound

Hay 2 tipos de cotas. Suponiendo problemas de minimización:

- **Cota Superior:** Coste de una solución válida que no debe sobrepasarse.
- **Cota Inferior:** Valor estimado de un nodo **n** del árbol, calculado como el coste real de viajar desde la raíz hasta **n** + estimación para llegar desde **n** hasta un nodo hoja.

La idea básica de un algoritmo B&B consiste en explorar antes los nodos más prometedores del árbol de estados, eliminando nodos no factibles o que tengan una cota inferior peor o igual que el coste de la mejor solución conocida al problema (cota superior). Se hace uso de una cola con prioridad para ordenar los nodos vivos de más prometedor a menos prometedor.

Plantilla general de un algoritmo Branch&Bound

Procedimiento BranchAndBound(Problema P)

C= Cola con prioridad vacía

MS= solución inicial al problema P // Cálculo de cota superior (trivial, greedy, etc.)

Crear r=nodo raíz del árbol de estados del problema P

Insertar r en C

Mientras C no esté vacía, **hacer**:

 x= Quitar Primer Elemento de C

Para cada hijo v posible de x **hacer**:

Si v es nodo hoja **entonces**: // Si v es solución factible

Si coste(v) mejor que coste(MS), **entonces**: // Actualizar cota superior
 actualizar MS= solución asociada al nodo v

 Eliminar de C todos los nodos c con $\text{cotaInferior}(c) \geq \text{coste}(MS)$

Fin-Si

En otro caso, **Si** $\text{CotaInferior}(v) < \text{coste}(MS)$, insertar v en C

Fin-Para

Fin-Mientras

Devolver MS



UNIVERSIDAD
DE GRANADA

Algoritmos para exploración en grafos

1. Introducción
2. Representación de espacios de estados
3. La técnica Backtracking
4. Las 8 reinas
5. Resolución de Sudokus
6. El viajante de comercio
7. La técnica de Branch&Bound
- » 8. Asignación de tareas
9. El viajante de comercio



DECSAI

Enunciado

Sean n tareas a repartir entre n personas. Suponiendo que el coste de que la persona i se asigne a la tarea j es c_{ij} . Asignar las tareas a las personas de modo que el coste total sea mínimo.

Ejemplo: con $n=3$

	T1	T2	T3
P1	4	7	3
P2	2	6	1
P3	3	9	4

Asignación óptima:

P1= T2

P2= T3

P3= T1

Coste= 11

Elementos Branch&Bound (I)

■ **Representación:** $T(x_1, x_2, \dots, x_n)$ es un vector donde cada componente i representa una persona y x_i es la tarea asignada a esa persona.

■ **Restricciones implícitas:** x_i tendrá valores entre 1 y n (inclusive), la tarea asignada a la persona i .

■ **Restricciones explícitas:** No puede haber 2 personas haciendo la misma tarea.

■ **Árbol del estados:**

■ **Nodo raíz:** No hay asignada ninguna persona a ninguna Tarea

■ **En cada nivel i del árbol se asigna una tarea a la persona i .**

■ **Nodo hoja:** Tiene todas las personas/tareas asignadas

■ **Función objetivo:** Encontrar una asignación óptima que haga mínimo el coste total.

Elementos Branch&Bound (II)

■ Cálculo de cotas:

■ Cota inicial (superior): Solución trivial asignando la persona i a la tarea i

■ Cota de nodos (inferior): Suponiendo escogidos las k -ésimas primeras personas y sus tareas asignadas, acotamos los restantes eligiendo para cada tarea el menor coste (cota mínima), **tanto si es una solución válida como si no**

■ **Poda:** Se calcula el coste de la asignación realizada hasta el nivel actual + la cota hasta llegar a la solución. Los nodos con cota mínima mayor que este coste, se podan.



Ejemplo: con $n=4$

	T1	T2	T3	T4
P1	11	12	18	40
P2	14	15	13	22
P3	11	17	19	23
P4	17	14	20	28

	T1	T2	T3	T4
P1	11	12	18	40
P2	14	15	13	22
P3	11	17	19	23
P4	17	14	20	28

Cota para la solución inicial (superior)

$P_i = i = 11 + 15 + 19 + 28 = 73$.

Si en alguna rama se estima una cota mínima superior a 73, se poda.

Estimación de cota (inferior) para la raíz:
Para cada tarea sin asignar, se calcula su mínimo coste: $11 + 12 + 13 + 22 = 55$

Lo mínimo posible a alcanzar es 55, aunque esta solución no sea factible por tener P1 y P2 asignadas varias tareas.

Ejemplo: con $n=4$

	T1	T2	T3	T4
P1	11	12	18	40
P2	14	15	13	22
P3	11	17	19	23
P4	17	14	20	28

Estimación de coste para $P1=1$

Para cada tarea sin asignar, se calcula su mínimo coste: $11+14+13+22=60$

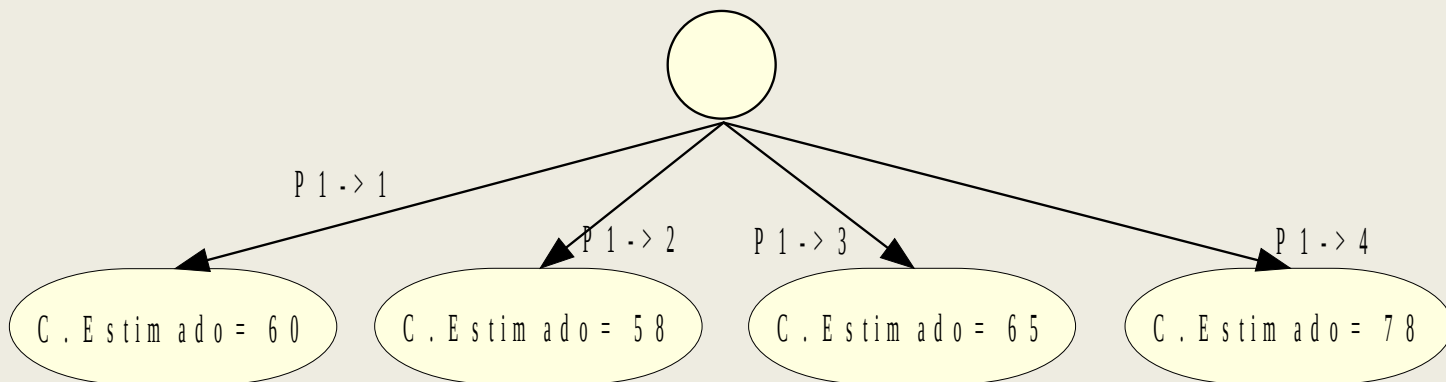
	T1	T2	T3	T4
P1	11	12	18	40
P2	14	15	13	22
P3	11	17	19	23
P4	17	14	20	28

Estimación de coste para $P1=2$

Para cada tarea sin asignar, se calcula su mínimo coste: $11+12+13+22=58$

Ejemplo: con $n=4$

■ Árbol de estados para asignación de P1.



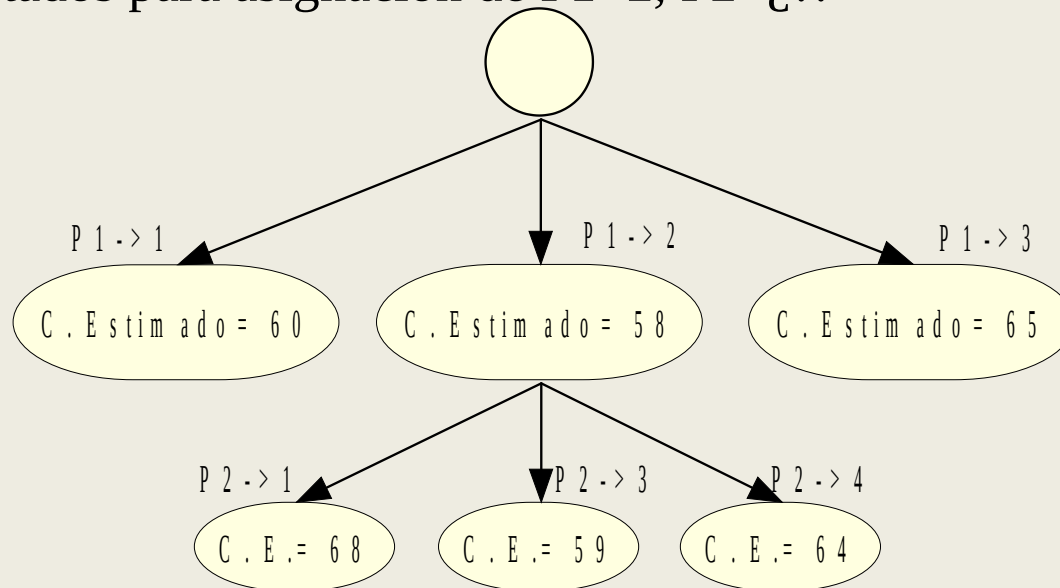
■ Se poda P1=4 porque es mayor que la cota, $78 > 73$.

■ Hacemos P1=2 nodo en expansión, porque es el más prometedor.

Ejemplo: con $n=4$

Expandimos nodo más prometedor.

Árbol de estados para asignación de $P1=2$, $P2=?$.

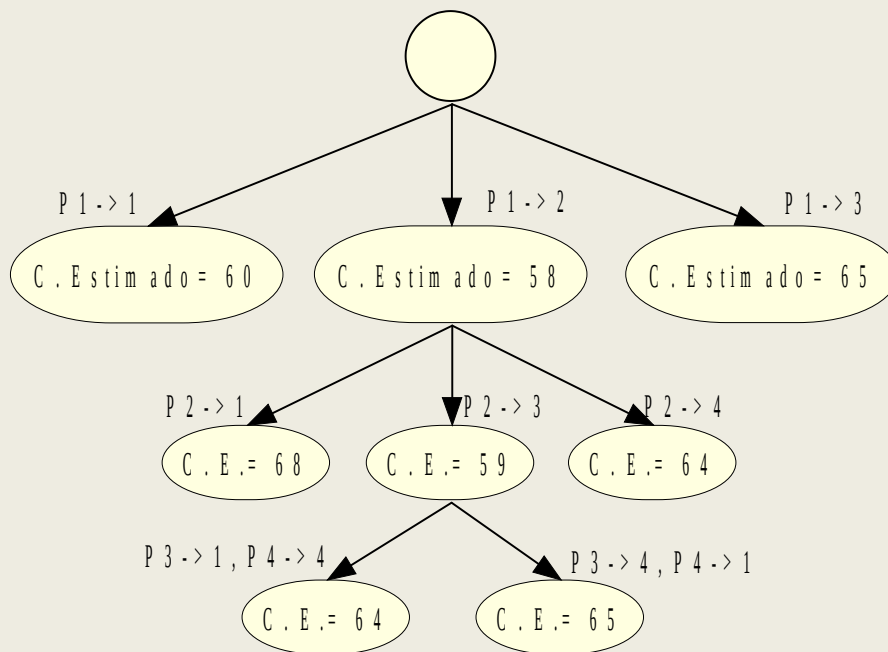


Expandimos nodo más prometedor, $P2=3$

Ejemplo: con $n=4$

Expandimos nodo más prometedor.

Árbol de estados para asignación de $P1=2$, $P2=3$

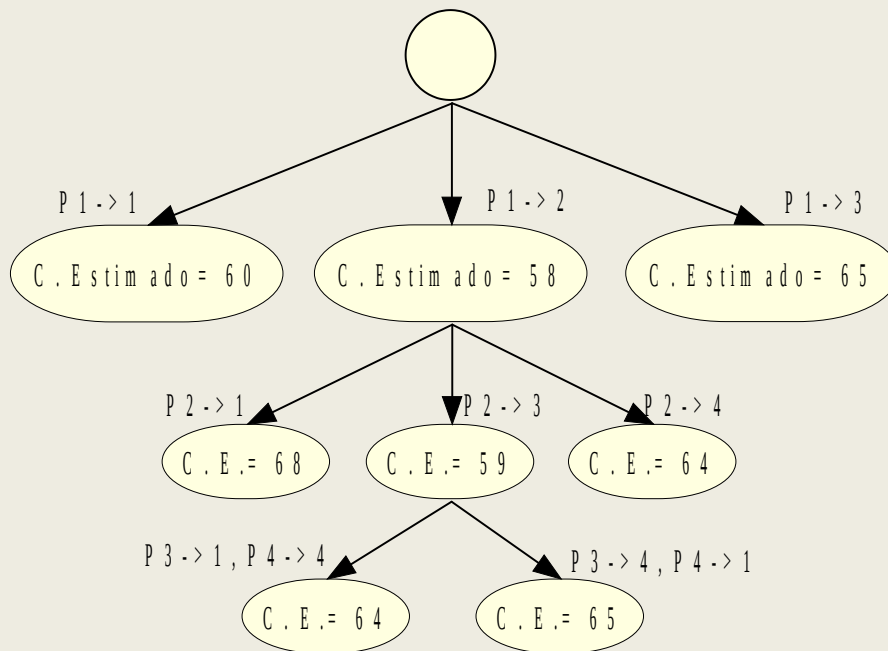


Actualizamos cota máxima= 64

Ejemplo: con $n=4$

Expandimos nodo más prometedor.

Árbol de estados para asignación de $P1=2$, $P2=3$

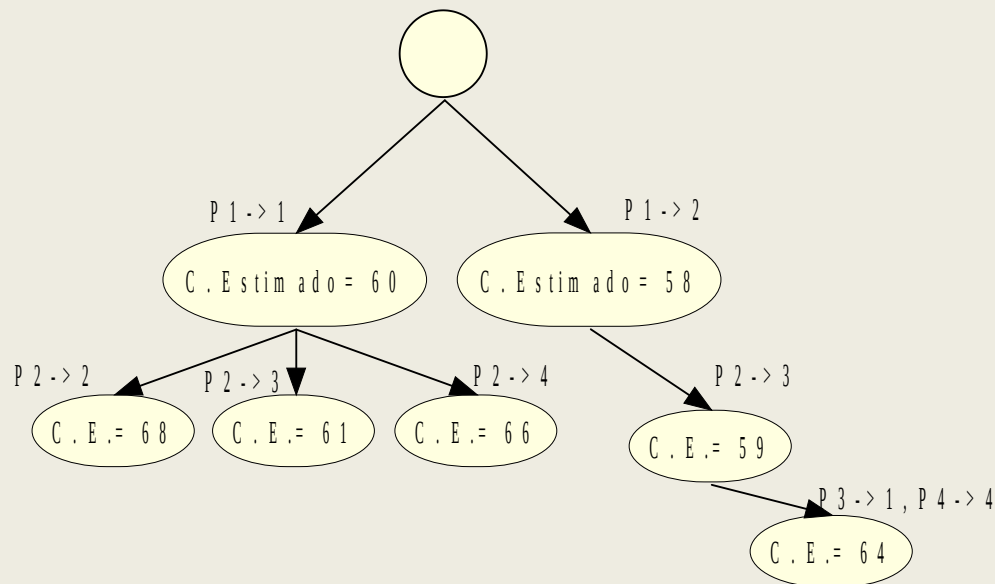


Actualizamos cota máxima = 64. Se podan los nodos con cota mínima igual o superior a dicha cota máxima.

Ejemplo: con $n=4$

Expandimos nodo vivo de menor coste: $P1=1$

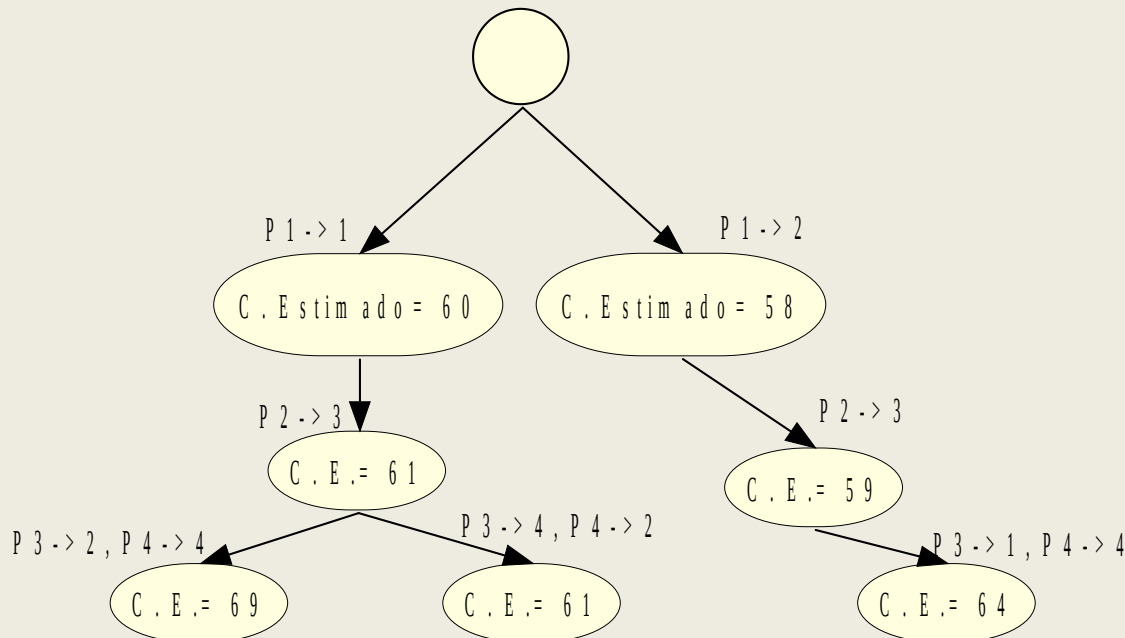
Árbol de estados para asignación de $P1=1$



Ejemplo: con $n=4$

■ Seleccionamos nodo en expansión $P2=3$

■ Árbol de estados para asignación de $P1=1$, $P2=3$



■ Actualizamos cota superior = 61. Podemos nodos con cota mínima ≥ 61 .

■ No quedan nodos vivos. Fin. Coste de la solución: **61**.



UNIVERSIDAD
DE GRANADA

Algoritmos para exploración en grafos

1. Introducción
2. Representación de espacios de estados
3. La técnica Backtracking
4. Las 8 reinas
5. Resolución de Sudokus
6. El viajante de comercio
7. La técnica de Branch&Bound
8. Asignación de tareas
- » 9. El viajante de comercio



DECSAI

Enunciado del problema

Sea un grafo $G=(V,A)$, **no dirigido**, **completo**, con V vértices y A aristas, donde las **aristas están ponderadas con pesos no negativos**.

El problema del viajante de comercio consiste en:

“Encontrar el circuito hamiltoniano minimal del grafo G .”

E
V
I
Z



PLUS

ULTRA

Elementos Branch&Bound (I)

■ **Representación:** $T(x_1, x_2, \dots, x_n)$ es un vector donde cada componente i representa una posición y x_i es el nodo que se visita en la i -ésima posición. El valor n =número de nodos del grafo.

■ **Restricciones implícitas:** x_i tendrá valores entre 1 y n (inclusive).

■ **Restricciones explícitas:** No puede haber un nodo visitado dos veces, $x_i \neq x_j$ para todo i, j

■ **Árbol del estados:**

■ **Nodo raíz:** Vector $T = \{1, 0, 0, \dots, 0\}_n$, Suponiendo que se parte de la ciudad 1

■ En cada nivel i del árbol se resuelve el nodo que se visita en la i -ésima posición.

■ **Nodo hoja:** Tiene todos los nodos asignados

Elementos Branch&Bound (II)

■ **Función objetivo:** Encontrar una ruta óptima para el circuito hamiltoniano que representa la solución

■ **Cálculo de cotas:**

■ **Cota inicial (superior):** Solución trivial asignando el nodo i en la i -ésima posición a visitar.

■ **Cota de nodos (inferior):** Suponiendo escogido el orden de visita de los nodos para las k -ésimas posiciones, acotamos las siguientes seleccionando el arco de valor mínimo entre los nodos que aún no han sido visitados, **tanto si es una solución válida como si no**

■ **Podar:** Se calcula el coste de la asignación realizada hasta el nivel actual + la cota hasta llegar a la solución. Los nodos con cota mínima mayor que este coste, se podan.



Ejercicio:

Realizar la ejecución del algoritmo diseñado para el grafo dado por la matriz de adyacencia de la diapositiva.

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



UNIVERSIDAD
DE GRANADA



Algorítmica

Grado en Ingeniería Informática

Tema 5 – Algoritmos para exploración en grafos

*Este documento está protegido por la
Ley de Propiedad Intelectual (
Real Decreto Ley 1/1996 de 12 de abril).
Queda expresamente prohibido su uso o
distribución sin autorización del autor.*

Manuel Pegalajar Cuéllar

manupc@ugr.es

Departamento de Ciencias de la
Computación e Inteligencia Artificial

<http://decsai.ugr.es>