Tema 4: Búsqueda informada mediante técnicas heurísticas

José Luis Ruiz Reina
José Antonio Alonso
Franciso J. Martín Mateos

[1]Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Inteligencia Artificial I, 2011

Índice

Búsqueda informada

- Búsqueda ciega o no informada (anchura, profundidad,...): no cuenta con ningún conocimiento sobre cómo llegar al objetivo.
- Búsqueda informada: aplicar *conocimiento* al proceso de búsqueda para hacerlo más eficiente.
- El conocimiento vendrá dado por una función que *estima* la "bondad" de los estados:
  - Dar preferencia a los estados mejores.
  - Ordenando la cola de **`ABIERTOS`**, comparando su bondad estimada.
  - Objetivo: reducir el árbol de búsqueda, ganando *eficiencia* en la práctica.

Concepto de heurística

- Heurística:
  - Del griego *heuriskein*, descubrir: *¡Eureka!*
  - Según el DRAE: “técnica de la indagación y del descubrimiento”.
  - Otro significado: método para resolver problemas que no garantiza la solución, pero que en general funciona bien.
  - En nuestro caso, una heurística será una función numérica sobre los estados.
- Función heurística, **`heurística(estado)`**:
  - Estima la “distancia” al objetivo.
  - Siempre mayor o igual que 0.
  - Valor en los estados finales: 0.
  - Admitimos valor ∞.
- Todo el conocimiento específico que se va a usar sobre el problema está codificado en la función heurística.

Recordar: problema del viaje por Andalucía

- Ir de Sevilla a Almería, mediante una secuencia de desplazamientos a capitales de provincia fronterizas
- 8 posibles estados:
  - Almería, Cádiz, Córdoba, Granada, Huelva, Jaen, Málaga, Sevilla
- Estado inicial: Sevilla.
- Estado final: Almeria.
- Operadores:
  - Ir a Almería, Ir a Cádiz, Ir a Córdoba, Ir a Granada, Ir a Huelva, Ir a Jaén, Ir a Málaga, Ir a Sevilla.

Una heurística para el problema del viaje

- Coordenadas:

```
Almeria : (409.5  93  )
Granada : (309   127.5)
Malaga  : (232.5  75  )
Cadiz   : ( 63    57  )
Huelva  : (  3   139.5)
Sevilla : ( 90   153  )
Cordoba : (198   207  )
Jaen    : (295.5 192  )
```

- Función heurística (distancia en línea recta):

```
heurística(estado) =
      distancia(coordenadas(estado),
                coordenadas(almeria))
```

Recordar: problema del 8-puzle

- Un tablero cuadrado (3x3) en el que hay situados 8 bloques cuadrados numerados (con lo cual se deja un hueco del tamaño de un bloque). Un bloque adyacente al hueco puede deslizarse hacia él. El juego consiste en transformar una posición inicial en la posición final mediante el deslizamiento de los bloques. En particular, consideramos el estado inicial y final siguientes:

2	8	3
1	6	4
7		5

Estado inicial

1	2	3
8		4
7	6	5

Estado final

- Estados: cada una de las posibles configuraciones del tablero
- Operadores: arriba, abajo, izquierda, derecha (movimientos del hueco)

Primera heurística en el problema del 8–puzle

- Problema del 8-puzle (primera heurística):
  - **`heurística(estado)`**: número de piezas descolocadas respecto de su posición en el estado final
- Ejemplo:

2	8	3
1	6	4
7		5

H = 4

1	2	3
8		4
7	6	5

H = 0

Segunda heurística en el problema del 8–puzle

- Problema del 8-puzle (segunda heurística):
  - **`heurística(estado)`**: suma de las distancias Manhattan de cada pieza a donde debería estar en el estado final
- Ejemplo:

2	8	3
1	6	4
7		5

H = 5

1	2	3
8		4
7	6	5

H = 0

Idea de la búsqueda por primero el mejor

- Búsqueda por primero el mejor:
  - Analizar preferentemente los nodos con heurística más baja.
  - Ordenar la cola de abiertos por heurística, de menor a mayor
- También llamada búsqueda *voraz* o *codiciosa* (del inglés *"greedy"*)
  - Porque siempre elige expandir *lo que estima* que está más "cerca" del objetivo
- Su rendimiento dependerá de la bondad de la heurística usada.

Implementación de búsqueda por primero el mejor

- Nodo de búsqueda: estado + camino + heurística
  - Funciones de acceso: **`ESTADO(NODO)`**, **`CAMINO(NODO)`** y **`HEURISTICA-DEL-NODO(nodo)`**
- Sucesores de un nodo heurístico:

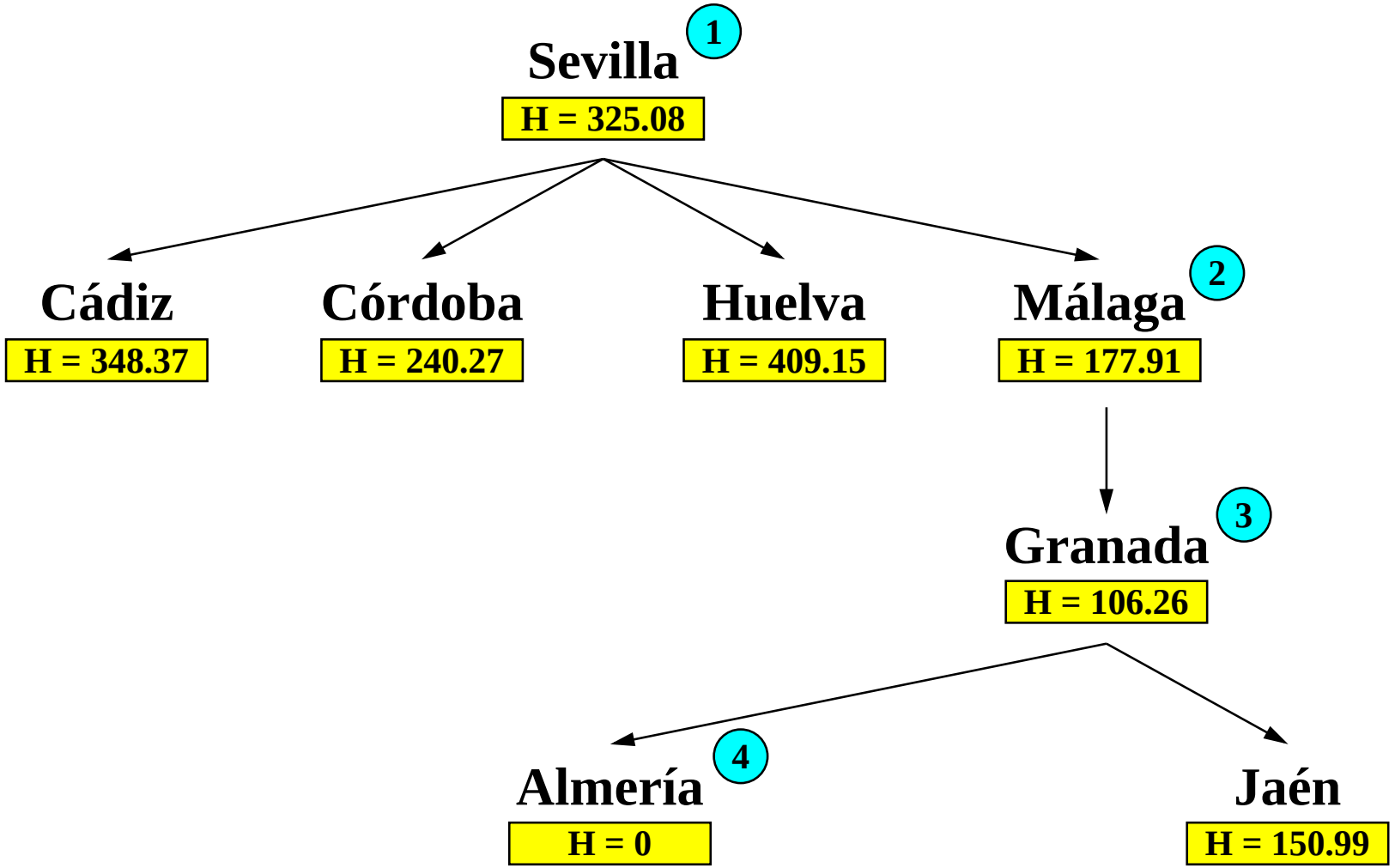
```
FUNCION SUCESOR(NODO,OPERADOR)
1. Hacer ESTADO-SUCESOR
    igual a APLICA(OPERADOR,ESTADO(NODO))
2. Si ESTADO-SUCESOR=NO-APLICABLE
      devolver NO-APLICABLE
   en caso contrario,
      devolver un nodo cuyo estado es ESTADO-SUCESOR, cuyo camino
       es el resultado de añadir OPERADOR a CAMINO(NODO) y cuya
       heurística es la de ESTADO-SUCESOR

FUNCION SUCESORES(NODO)
1. Hacer SUCESORES vacío
2. Para cada OPERADOR en *OPERADORES*,
       si SUCESOR(NODO,OPERADOR) =/= NO-APLICABLE,
          incluir SUCESOR(NODO,OPERADOR) en SUCESORES
3. Devolver SUCESORES
```

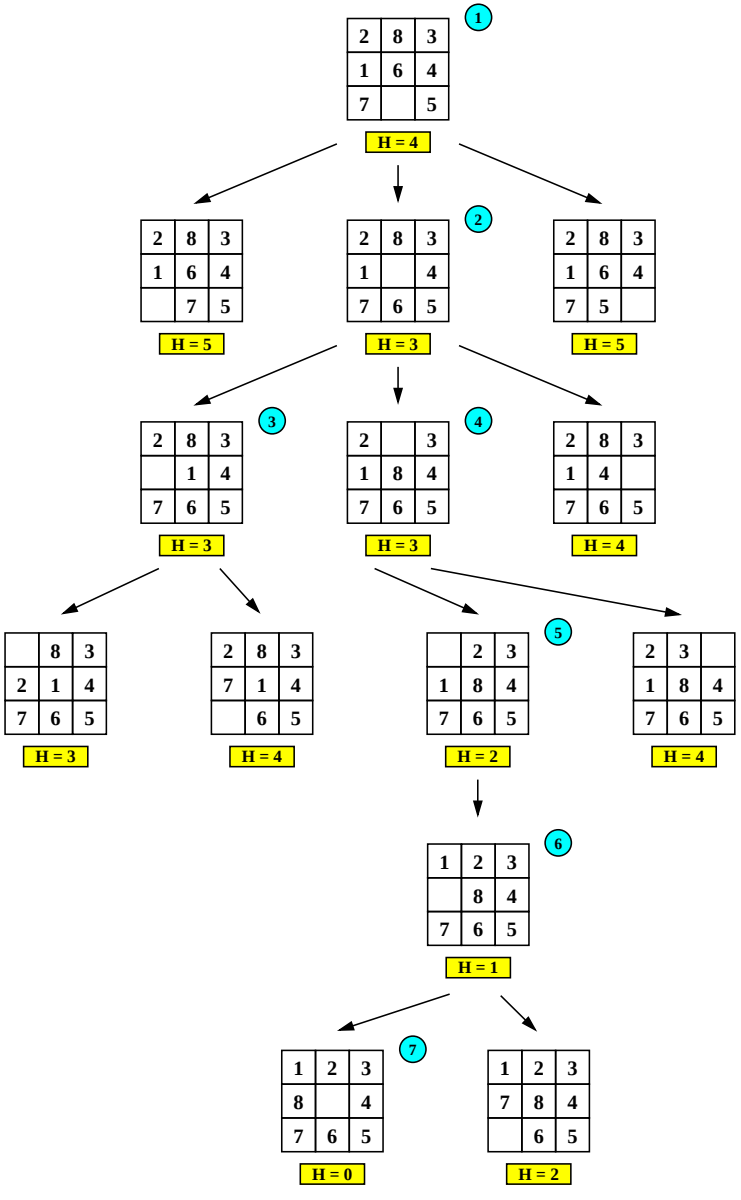
Implementación de la búsqueda por primero el mejor

```
FUNCION BUSQUEDA-POR-PRIMERO-EL-MEJOR()
1. Hacer ABIERTOS la cola formada por el nodo inicial (el nodo
    cuyo estado es *ESTADO-INICIAL*, cuyo camino es vacío
    y cuya heurística es la de *ESTADO-INICIAL*);
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
   2.1 Hacer ACTUAL el primer nodo de ABIERTOS
   2.2 Hacer ABIERTOS el resto de ABIERTOS
   2.3 Poner el nodo ACTUAL en CERRADOS.
   2.4 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
       2.4.1 devolver el nodo ACTUAL y terminar.
       2.4.2 en caso contrario,
             2.4.2.1 Hacer NUEVOS-SUCESORES la lista de nodos
                     de SUCESORES(ACTUAL) cuyo estado no está
                     ni en ABIERTOS ni en CERRADOS
             2.4.2.2 Hacer ABIERTOS el resultado de incluir
                     NUEVOS-SUCESORES en ABIERTOS y ordenar
                     en orden creciente de sus heurísticas
3. Devolver FALLO.
```

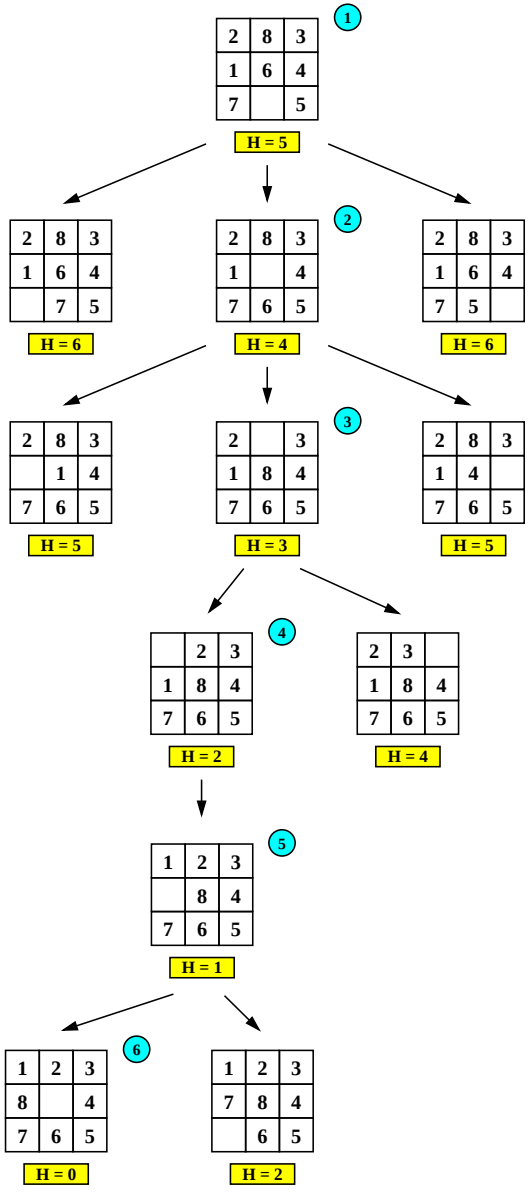
Primero el mejor para el problema del viaje



8–puzle por primero el mejor: heurística 1



8–puzle por primero-el-mejor: heurística 2



Propiedades de la búsqueda por primero el mejor

- Complejidad en tiempo $O(r^m)$, donde:
  - r: factor de ramificación.
  - m: profundidad máxima del árbol de búsqueda.
- Complejidad en espacio: $O(r^m)$.
- En la práctica, el tiempo y espacio necesario depende del problema concreto y de la calidad de la heurística usada
- No es completa, en general.
  - Por ejemplo, una mala heurística podría hacer que se tomara un camino infinito.
- No es minimal (no garantiza soluciones con el menor número de operadores).
  - La heurística podría guiar hacia una solución no minimal

Espacio de estados con coste

- En algunos problemas, existe un coste (positivo) asociado a la aplicación de los operadores.
  - **`coste-de-aplicar-operador(estado,operador)`**.
  - Asumimos que **`operador`** es aplicable a **`estado`**.
- Coste asociado a un camino:
  - Suma de los costes de la aplicación de cada uno de sus operadores.
  - Una *solución óptima* es una solución con coste mínimo.
- En algunos problemas, no existe un coste explícito
  - En ese caso, **`coste-de-aplicar-operador(estado,operador)=1`**, y las soluciones óptimas son las minimales.
- Coste en el problema del viaje: distancia euclídea entre **`estado`** y **`aplica(operador,estado)`**

Idea de la búsqueda óptima

- Analizar primero los nodos con menor coste.
- Es decir, ordenar la cola de abiertos por coste, de menor a mayor
- De esta manera, cuando se llega por primera vez a un estado, se llega con el menor coste posible.
  - Y en particular, la primera vez que se llega a un estado final tenemos una solución óptima
- Se trata de una *búsqueda ciega* o *no informada*:
  - No usa conocimiento para guiar la búsqueda hacia el objetivo
  - Caso particular: búsqueda en anchura.

Implementación de la búsqueda óptima

- Nodo de búsqueda: estado + camino + coste del camino
  - Funciones de acceso: **`ESTADO(NODO)`**, **`CAMINO(NODO)`** y **`COSTE-CAMINO(nodo)`**
- Sucesores de un nodo con coste:

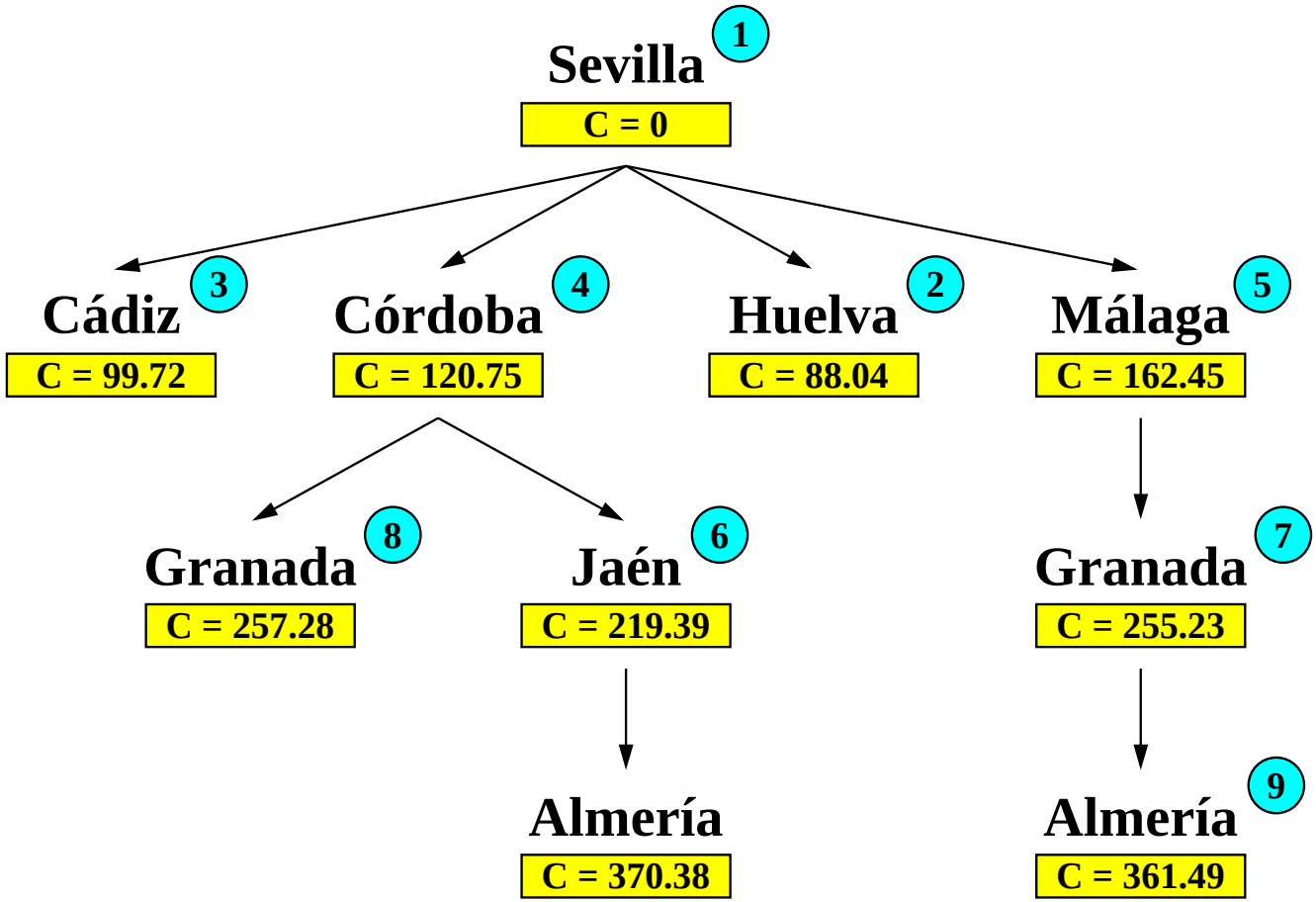
```
FUNCION SUCESOR(NODO,OPERADOR)
1. Hacer ESTADO-SUCESOR igual a APLICA(OPERADOR,ESTADO(NODO))
2. Si ESTADO-SUCESOR=NO-APLICABLE
    devolver NO-APLICABLE
   en caso contrario,
    devolver un nodo cuyo estado es ESTADO-SUCESOR, cuyo camino
    es el resultado de añadir OPERADOR a CAMINO(NODO) y cuyo
    coste es COSTE-CAMINO(CAMINO(NODO)) más
    COSTE-DE-APLICAR-OPERADOR(ESTADO(NODO),OPERADOR)

FUNCION SUCESORES(NODO)
1. Hacer SUCESORES vacío
2. Para cada OPERADOR en *OPERADORES*,
       si SUCESOR(NODO,OPERADOR) =/= NO-APLICABLE,
          incluir SUCESOR(NODO,OPERADOR) en SUCESORES
3. Devolver SUCESORES
```

Implementación de la búsqueda óptima

```
FUNCION BUSQUEDA-OPTIMA()
1. Hacer ABIERTOS la cola formada por el nodo inicial (el nodo con
         estado *ESTADO-INICIAL*, camino vacío y coste 0);
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
   2.1 Hacer ACTUAL el primero de ABIERTOS
       Hacer ABIERTOS el resto de ABIERTOS
   2.2 Poner el nodo ACTUAL en CERRADOS.
   2.3 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
       2.3.1 devolver el nodo ACTUAL y terminar.
       2.3.2 en caso contrario,
             2.3.2.1 Hacer NUEVOS-SUCESORES la lista de nodos de
                     SUCESORES(ACTUAL) que o bien tienen un estado
                     que no aparece en los nodos de ABIERTOS ni de
                     CERRADOS, o bien su coste es menor que cualquier
                     otro nodo con el mismo estado que apareciera
                     en ABIERTOS o en CERRADOS
             2.3.2.2 Hacer ABIERTOS el resultado de incluir
                     NUEVOS-SUCESORES en ABIERTOS y ordenar
                     todo en orden creciente de los costes
                     de los caminos de los nodos
3. Devolver FALLO.
```

Árbol de búsqueda óptima para el problema del viaje

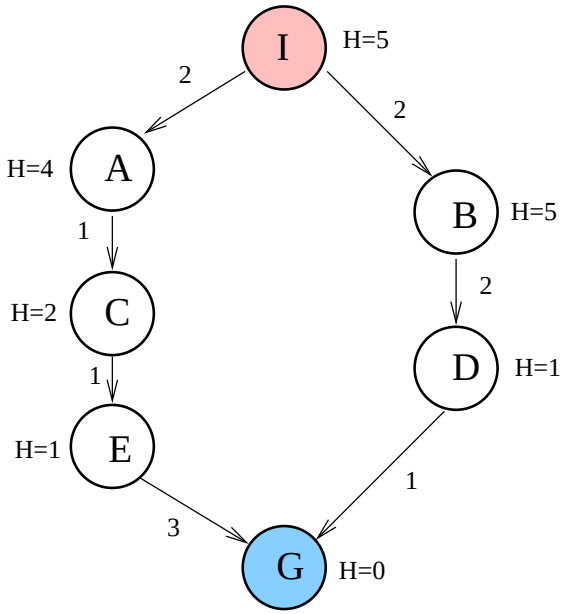


Propiedades de la búsqueda óptima

- Es completa (siempre que todos los costes individuales sean mayores que un cierto $\epsilon > 0$)
- Complejidad en tiempo y espacio: $O(r^{1+\lfloor C/\epsilon \rfloor})$, donde C es el coste de una solución óptima.
  - Exponencial (es una generalización de la búsqueda en anchura)
- Siempre encuentra solución óptima.
- Salvo en espacios de estados pequeños, en la práctica esta búsqueda no es posible, debido a la cantidad de tiempo y espacio que necesita

Uso de heurísticas para buscar soluciones óptimas

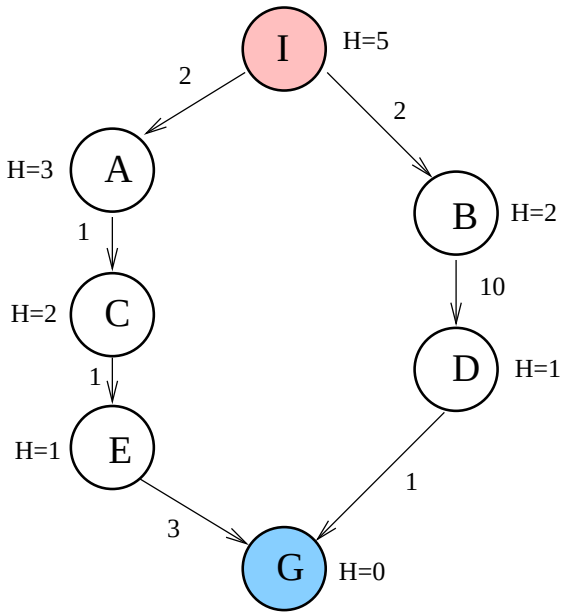
- ¿Podemos usar búsqueda por primero el mejor para acelerar la búsqueda y aún estar seguro de obtener soluciones óptimas? En general, NO.
- Ejemplo 1:



  - Solución encontrada por primero el mejor: I-A-C-E-G (subóptima)
  - Causa: la heurística *sobrestima* el coste real

Uso de heurísticas para buscar soluciones óptimas

- Ejemplo 2:



  - Solución encontrada por primero el mejor: I-B-D-G (subóptima)
  - Causa: no se han tenido en cuenta *los costes* de los caminos ya recorridos

Idea de la búsqueda A*

- Objetivo de la búsqueda A*:
  - conseguir buenas soluciones (óptimas).
  - ganar en eficiencia (reduciendo el árbol de búsqueda).
- Idea: asignar a cada nodo n un valor $f(n) = g(n) + h(n)$
  - $g(n)$: coste del camino hasta n
  - $h(n)$: heurística del nodo, *estimación* del coste *de un camino óptimo* desde n hasta un estado final
  - $f(n)$: estimación del coste total de una solución óptima *que pasa por* n
- Seleccionar siempre el nodo con menor valor de f
  - ordenando la cola de **`ABIERTOS`** en orden creciente respecto a f

Implementación de A*

- Nodo de búsqueda: estado + camino + coste del camino + coste-más-heurística
  - Funciones de acceso: **`ESTADO(NODO)`**, **`CAMINO(NODO)`**, **`COSTE-CAMINO(nodo)`** y **`COSTE-MAS-HEURISTICA(nodo)`**.
- Sucesores de un nodo con coste y heurística:

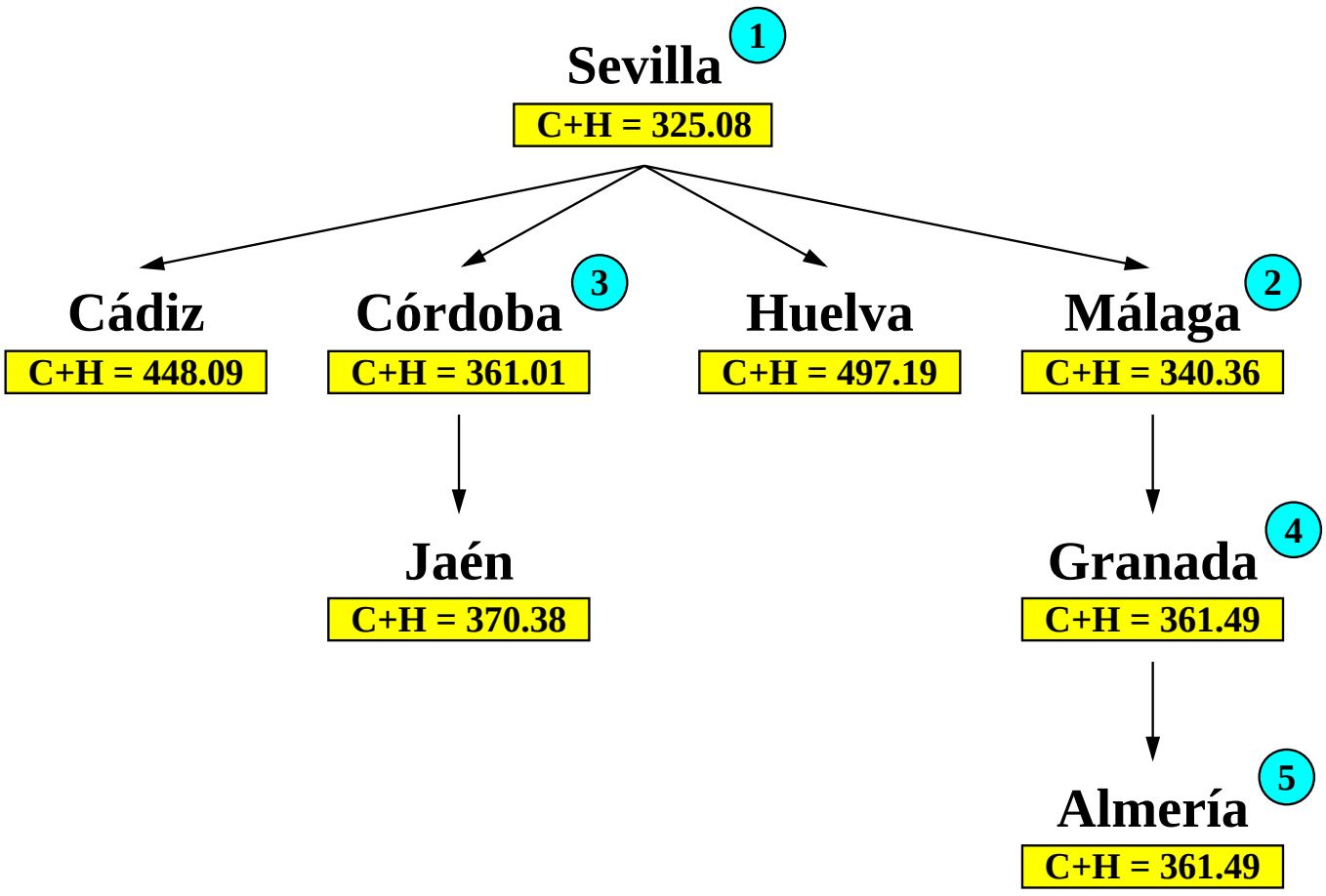
```
FUNCION SUCESOR(NODO,OPERADOR)
1. Hacer ESTADO-SUCESOR igual a APLICA(OPERADOR,ESTADO(NODO))
2. Si ESTADO-SUCESOR=NO-APLICABLE
    devolver NO-APLICABLE
   en caso contrario,
    devolver un nodo cuyo estado es ESTADO-SUCESOR, cuyo camino
     es el resultado de añadir OPERADOR a CAMINO(NODO) y cuyo
     coste es COSTE-CAMINO(CAMINO(NODO)) más
     COSTE-DE-APLICAR-OPERADOR(ESTADO(NODO),OPERADOR)
     y cuyo coste-más-heurística es el coste anterior
     más HEURISTICA(ESTADO(NODO))
```

- La función **`SUCESORES(NODO)`**, como anteriormente

Implementación de la búsqueda A*

```
FUNCION BUSQUEDA-A-ESTRELLA()
1. Hacer ABIERTOS la cola formada por el nodo inicial (el nodo cuyo
    estado es *ESTADO-INICIAL*, cuyo camino es vacío, cuyo coste
    es 0, y cuyo coste-más-heurística es HEURISTICA(*ESTADO-INICIAL*))
   Hacer CERRADOS vacío
2. Mientras que ABIERTOS no esté vacía,
   2.1 Hacer ACTUAL el primer nodo de ABIERTOS
       Hacer ABIERTOS el resto de ABIERTOS
   2.2 Poner el nodo ACTUAL en CERRADOS.
   2.3 Si ES-ESTADO-FINAL(ESTADO(ACTUAL)),
       2.3.1 devolver el nodo ACTUAL y terminar.
       2.3.2 en caso contrario,
             2.3.2.1 Hacer NUEVOS-SUCESORES la lista de nodos de
                     SUCESORES(ACTUAL) que o bien tienen un estado
                     que no aparece en los nodos de ABIERTOS ni
                     de CERRADOS, o bien su coste es menor que
                     cualquier otro nodo con el mismo estado que
                     apareciera en ABIERTOS o en CERRADOS
             2.3.2.2 Hacer ABIERTOS el resultado de incluir NUEVOS-SUC
                     en ABIERTOS y ordenar todo en orden creciente de
                     del coste-mas-heurística de los nodos
3. Devolver FALLO.
```

Árbol de búsqueda A* para el viaje



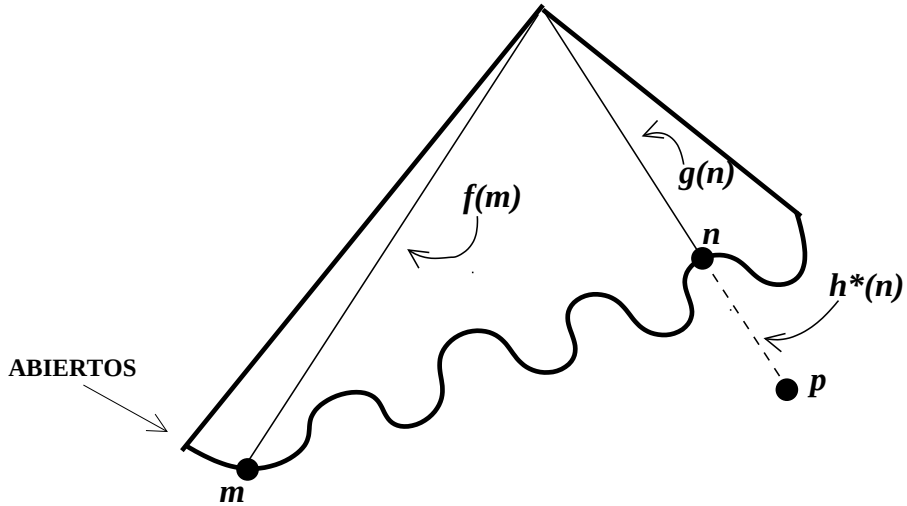
Propiedades de A*

- Sea $h^*(n)$ coste de un camino óptimo desde el estado de n hasta un estado final
  - $f^*(n) = g(n) + h^*(n)$ coste total de una solución óptima *que pasa por* n
- En la práctica, no conocemos h^*
  - Usamos una función heurística h que estima h^*
- Posibilidades:
  - Para todo nodo n, $h(n) = 0$: búsqueda óptima, no hay reducción del árbol de búsqueda
  - Para todo nodo n, $h(n) = h^*(n)$: estimación perfecta, no hay búsqueda
  - Para todo nodo n, $0 \leq h(n) \leq h^*(n)$: heurística *admisible*, solución óptima garantizada
  - Para algún nodo n, $h(n) > h^*(n)$: no se puede garantizar que la solución encontrada sea óptima

Propiedades de A*

- Usando una heurística admisible:
  - Es completa.
  - Encuentra siempre solución óptima.
- Complejidad en tiempo y en espacio: en el peor de los casos, como la búsqueda óptima
- En la práctica, el coste en tiempo y espacio depende del problema particular y de la calidad de la heurística usada

Demostración de que A* es óptimo



Supongamos que el algoritmo A* usa una heurística h admisible, y veamos que un nodo m correspondiente a una solución no óptima nunca puede estar el primero de la cola de **ABIERTOS**.

Sea p es un nodo correspondiente a una solución óptima. Se puede demostrar entonces que existe un nodo n en la lista de **ABIERTOS** cuyo camino es un subcamino del camino de p. Bastará probar que n está antes que m en la cola de **ABIERTOS** (es decir, que $f(n) < f(m)$).

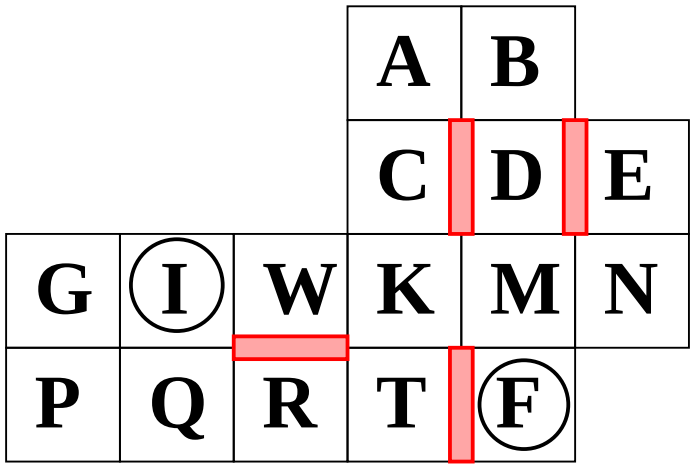
Por definición, $f(n) = g(n) + h(n)$. Como h es admisible, $h(n) \leq h^*(n)$. Como n y p están en el mismo camino a la solución óptima, por definición de g y de h^*, $g(n) + h^*(n) = g(p)$. Luego $f(n) \leq g(p)$. Puesto que m es subóptimo y $h(m) = 0$ (ya que m es final), se tiene que $g(p) < g(m) = f(m)$. En conclusión $f(n) < f(m)$.

Invención de heurísticas

- Recordar: estimación del coste mínimo restante hasta un estado final
- Comparación de heurísticas:
  - Si para todo nodo n, $0 \leq h_1(n) \leq h_2(n) \leq h^*(n)$, entonces h_2 está *más informada* que h_1 y mejora la eficiencia
- Técnicas para encontrar heurísticas
  - Relajación del problema
  - Combinación de heurísticas admisibles
  - Uso de información estadística
- Evaluación eficiente de la función heurística

Problema del laberinto

- En el siguiente laberinto, se puede pasar desde una casilla a otra de las posibles adyacentes (arriba, abajo, izquierda, derecha), salvo si existe una barrera entre ellas

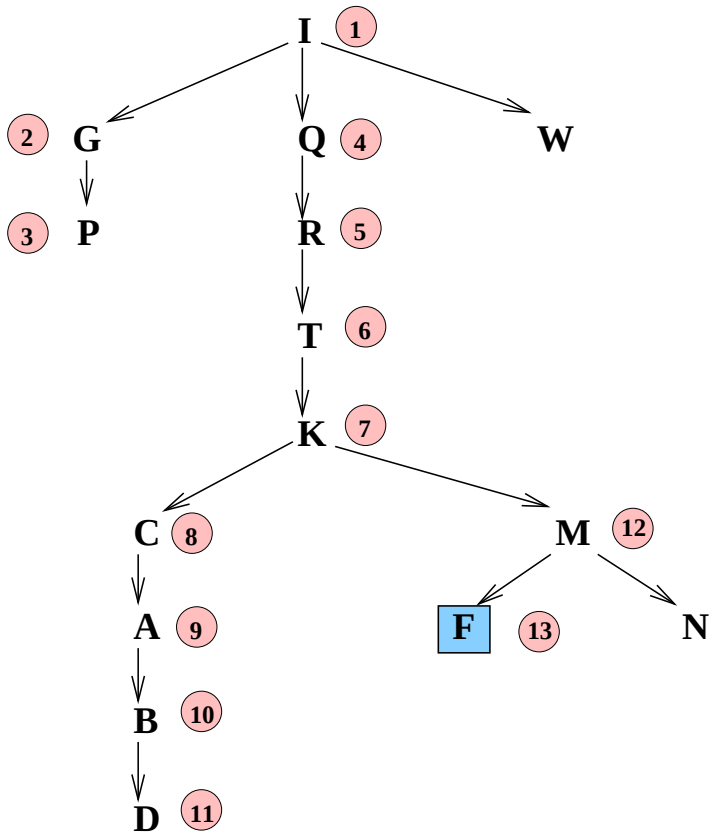


- Objetivo: ir de I a F
- Como ejercicio final, resolveremos el problema aplicando las búsquedas en profundidad, primero el mejor y A*

Problema del laberinto como problema de espacio de estados

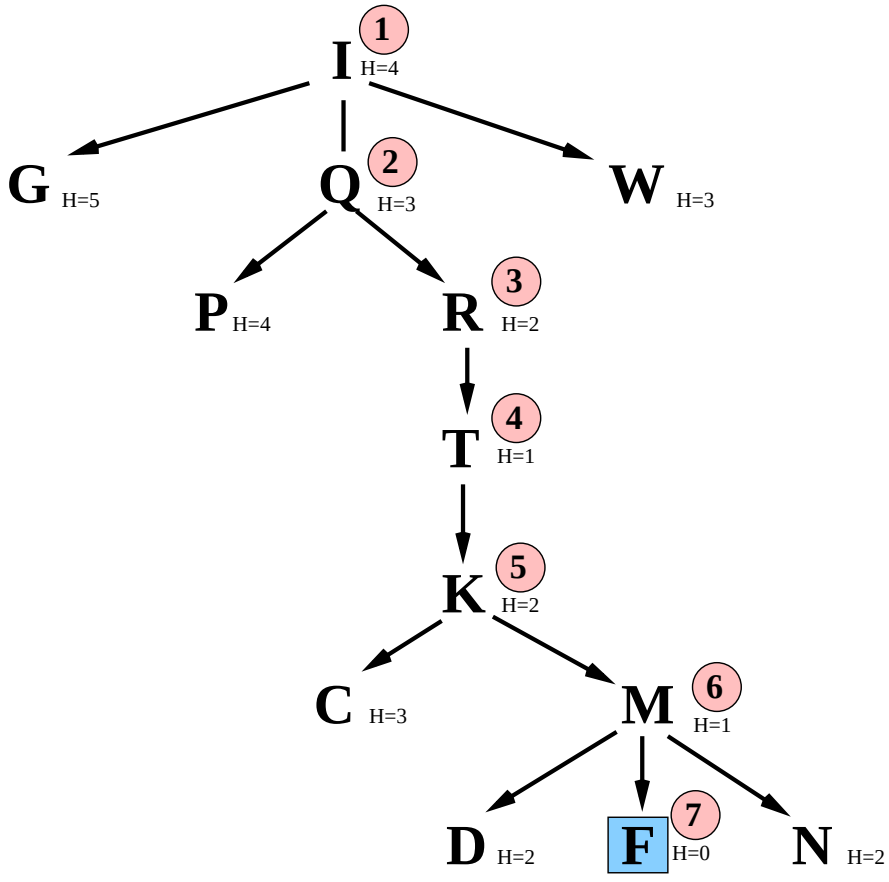
- Estados: A, B, C, D, E, G, I, W, K, M, N, P, Q, R, T y F
- Estado inicial: I
- Estado final: F
- Operadores: arriba, abajo, izquierda, derecha
- Aplicabilidad y resultado de la aplicación
  - “Arriba” es aplicable a un estado si existe una casilla arriba y no está separada por barrera; el resultado de aplicarlo es la casilla de arriba y su coste es 1
  - El resto de operadores, análogamente
  - Consideraremos que los sucesores están ordenados alfabéticamente
- Heurística (admisible): “distancia Manhattan del estado a la casilla F”
- Durante la búsqueda, y con igual valoración, elegir por orden alfabético

Laberinto: búsqueda en profundidad



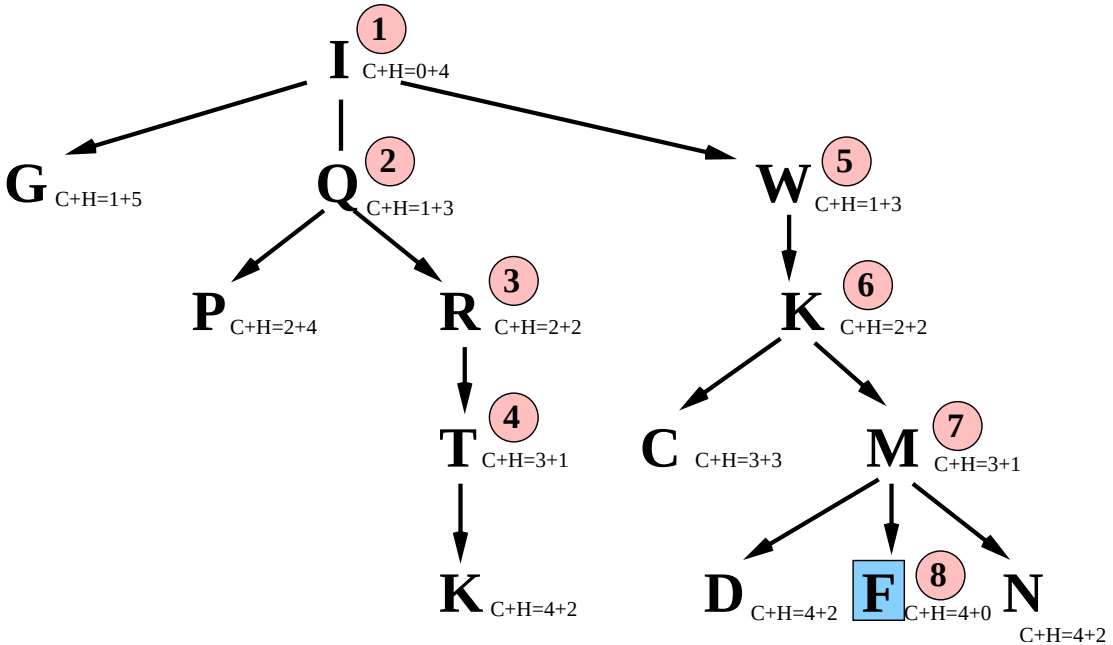
- Solución encontrada: I-Q-R-T-K-M-F
  - No es óptima
  - Nodos analizados: 13

Laberinto: búsqueda primero el mejor



- Solución encontrada: I-Q-R-T-K-M-F
  - No es óptima
  - Nodos analizados: 7
  - La heurística ha servido para reducir el espacio de búsqueda

Laberinto: búsqueda A*



- Solución encontrada: I-W-K-M-F
  - Óptima (heurística admisible)
  - Nodos analizados: 8
  - Aunque en este caso se analizan más nodos (se ha explorado parcialmente un camino equivocado), la solución óptima está asegurada
  - Nótese que se generan dos nodos distintos con el mismo estado K (pero distinto camino)

Bibliografía

- Russell, S. y Norvig, P. *Inteligencia artificial: Un enfoque moderno (segunda edición)* (Prentice Hall, 2004).
  - Cap. 3: “Solución de problemas mediante búsqueda”
- Russell, S. y Norvig, P. *Artificial Intelligence (A Modern Approach)* (Prentice–Hall, 2010). Third Edition
  - Cap. 3: “Solving problems by searching”.

Bibliografía

- Luger, G.F. *Artificial Intelligence (Structures and Strategies for Complex Problem Solving (4 edition)* (Addison–Wesley, 2002)
  - Cap. 4: “Heuristic search”
- Nilsson, N.J. *Inteligencia artificial (Una nueva síntesis)* (McGraw–Hill, 2001)]
  - Cap. 8: “Búsqueda heurística”
- Poole, D.; Mackworth, A. y Goebel, R. *Computational Intelligence (A Logical Approach)* (Oxford University Press, 1998).
  - Cap. 4: “Searching”