Bloque 1 – Representación de Conocimiento y Búsqueda

Tema 5:
Búsqueda Heurística

Bloque 1, Tema 5- Índice

Bibliografía

- S. Russell, P. Norvig. ***Artificial Intelligence. A modern approach.*** Prentice Hall, 3rd edición, 2010 (Capítulo 3) http://aima.cs.berkeley.edu/

Alternativamente:

- S. Russell, P. Norvig. ***Inteligencia artificial . Una aproximación moderna.*** Prentice Hall, 2ª edición, 2004 (Capítulos 3 y 4) http://aima.cs.berkeley.edu/2nd-ed/

Búsqueda en IA

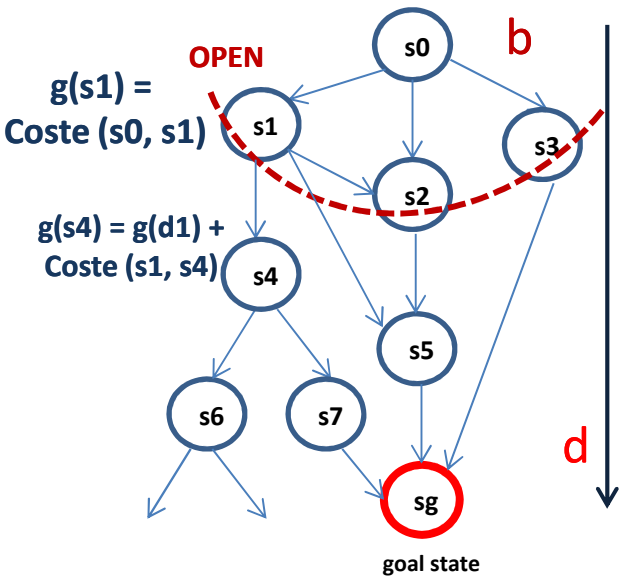
Objetivo: Obtener senda de mínimo coste desde raíz a un estado meta.

a) **Búsqueda en Árbol**: Solo lista OPEN (frontera), ordenados según f(n).
Expansión de un nodo: se extrae de OPEN y sucesores se añaden a OPEN.
  a) Sin control repetidos: nodos repetidos se repiten (como si fueran nuevos)
  b) Con control repetidos *(solo interesa mejor solución):* Nodo repetido en OPEN reemplaza al viejo si mejora su f(n). Si no, se olvida.

- Se pueden generar ciclos y caminos redundantes (con nodos ya expandidos)!

b) Búsqueda en Grafo. Lista OPEN (frontera) y CLOSED (nodos expandidos).
Expansión de un nodo: se extrae de OPEN y se añade a CLOSED.
a) Nodo generado es nuevo (no en OPEN ni en CLOSED): se añade a OPEN
b) Nodo repetido en OPEN: Reemplaza al viejo si mejora su f(n). Si no, se olvida.
c) Nodo repetido en CLOSED: Si mejora f(n) se re-expande: Se inserta en OPEN. Si no, se olvida. *(Si h(n) consistente, nunca mejorará f(n).)*

- No se generan caminos redundantes ni ciclos.



f(n) ⇒	ANCHURA	C. UNIFOR.	PROFUND	PROF. ITER.
CRITERIO	f(n)=Nivel(n)	f(n)=g(n)	- Nivel	Prof + Anch
COMPLETA	SI	SI	NO	SI
ÓPTIMA	SI*	SI	NO	SI*
C. ESPACIAL	$O(b^d)$	$\approx O(b^d)$	O(b m)	O(b d)
C. TEMPORAL	$O(b^d)$	$\approx O(b^d)$	$O(b^m)$	$O(b^d)$

- **Información de cada nodo:** **g(n)**:coste desde raíz
- **Estrategia de Búsqueda:** Ordenación nodos frontera en **OPEN**, según f(n)
- **Prueba de objetivo:** Cuando el nodo se selecciona para expandirse.
- Ramificación **(b)**, Nivel Solución **(d)**

1. Búsqueda heurística: f(n) en OPEN

Búsqueda heurística o informada: utiliza conocimiento específico del problema para guiar la búsqueda.

Suele encontrar soluciones más eficientemente que una búsqueda no informada. Es especialmente útil en problemas complejos de explosión combinatoria (p. ej.: problema de viajante).

¿Por qué utilizar heurísticas? En ocasiones no es viable utilizar una búsqueda sistemática que garantice optimalidad. La utilización de algunas heurísticas permiten obtener una *buena* solución aunque no sea la óptima.

Guía inteligente del proceso de búsqueda que permite podar grandes partes del árbol de búsqueda.

¿Por qué utilizar heurísticas es apropiado?

1. Normalmente, en problemas complejos, no necesitamos soluciones óptimas, una *buena* solución es suficiente.
2. La solución de la heurística para el caso peor puede no ser muy buena, pero en el mundo real el caso peor es poco frecuente.
3. Comprender el por qué (por qué no) funciona una heurística ayuda a profundizar en la compresión del problema

1. Búsqueda heurística

Aproximación general búsqueda primero-el-mejor:

- Proceso de búsqueda en árbol o grafo (TREE-SEARCH ó GRAPH-SEARCH) en el que un nodo se selecciona para ser expandido en base a una función de evaluación f(n)
- f(n) es una estimación de coste de modo que el nodo con el coste más bajo se expande primero de la cola de prioridades
- Todas las estrategias de búsqueda se pueden implementar usando f(n); la elección de f determina la estrategia de búsqueda
- Dos casos especiales de búsqueda primero el mejor: *búsqueda voraz* y *búsqueda A**.

TREE-SEARCH (OPEN) ó GRAPH-SEARCH (OPEN y CLOSED)

Proceso general de búsqueda

1. *nodo-actual <- Estado inicial del problema*
2. *Comprobar si nodo-actual es el estado final del problema; en dicho caso, FIN.*
3. *Aplicar las acciones del problema en dicho estado y generar el conjunto de nuevos estados.*
4. *Escoger un nodo que no ha sido expandido todavía (nodo-actual)*
5. *Ir al paso 2*

2. Búsqueda *voraz*

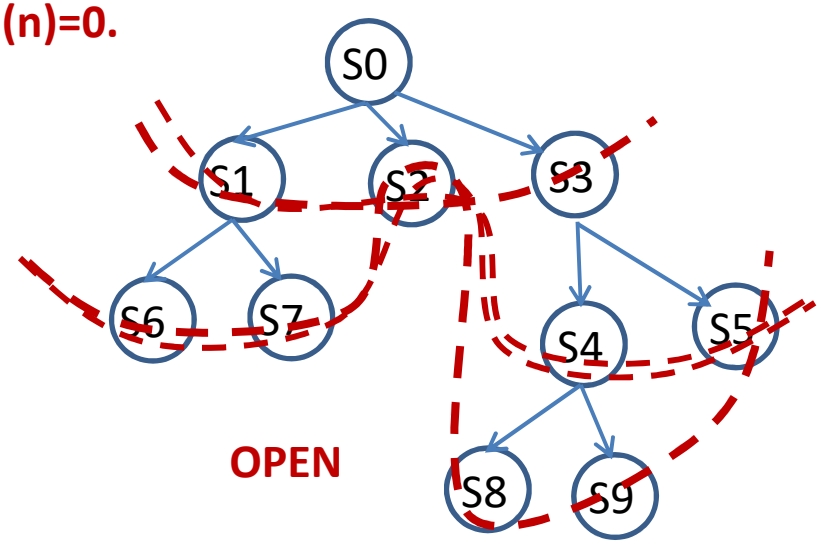
La mayoría de algoritmos primero-el-mejor incluyen una función heurística h(n) como componente de la función f.

Función heurística: *Simplificación o conjetura que reduce o limita la búsqueda de soluciones en dominios complejos o poco conocidos.*

h(n)= *coste estimado* del camino óptimo desde el estado representado en el nodo n al estado de objetivo.

Si n es el estado de objetivo entonces h(n)=0.

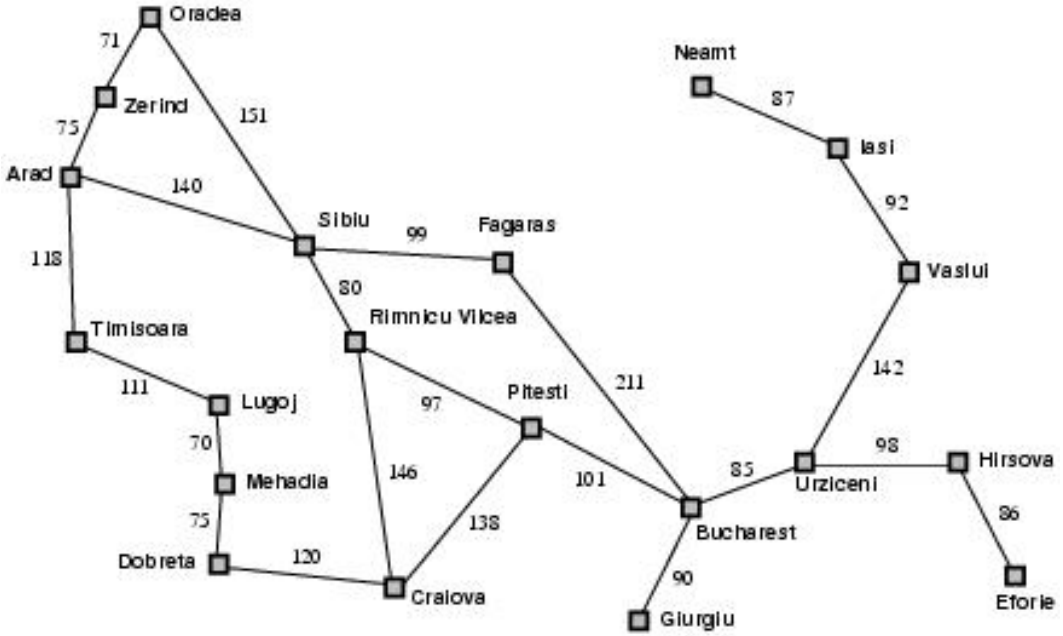
La búsqueda *voraz* expande el nodo que parece estar más cerca del objetivo ya que, problamente, dicho nodo conduce más rápidamente a una solución.



Búsqueda Voraz:

Evalúa nodos utilizando simplemente: **f(n)=h(n).**

2. Búsqueda voraz: el ejemplo de Rumanía



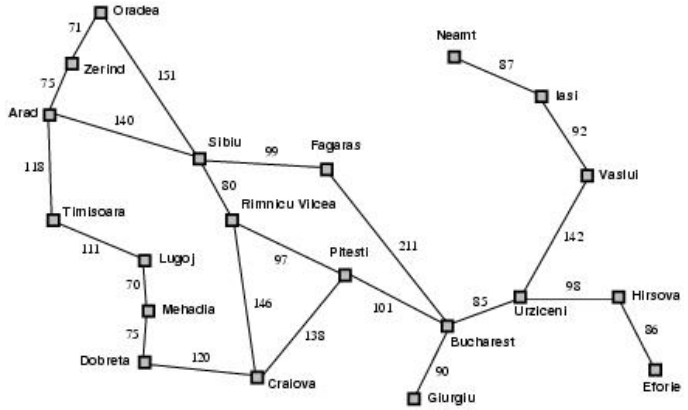
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h_{SLD} : heurística 'distancia en línea recta' de una ciudad a Bucarest

Ejemplos:

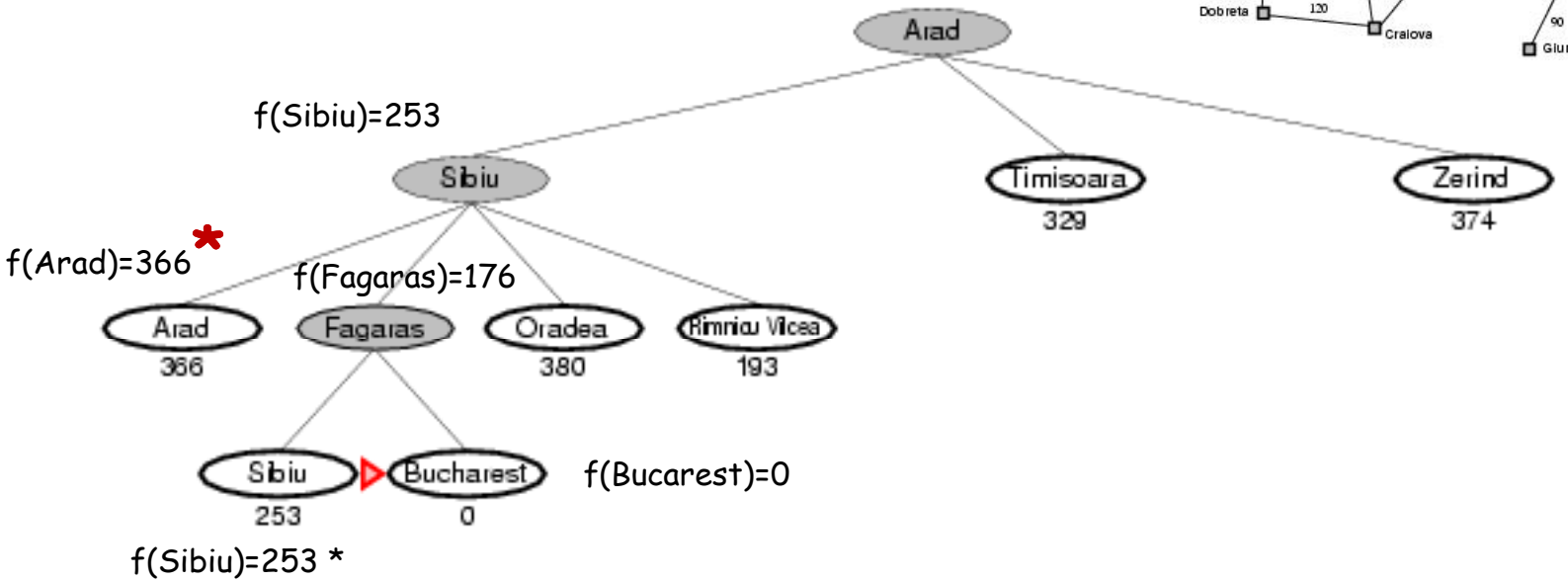
h(*Arad*)=366
h(Fagaras)=176
h(Bucarest)=0

2. Búsqueda voraz: el ejemplo de Rumanía



f(n)=h(n)

- Expande el nodo más cercano al objetivo
- Búsqueda primero-el-mejor voraz



Objetivo alcanzado: solución no óptima

(ver solución alternativa: Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucarest)

* En la versión GRAPH-SEARCH se comprobarían estados repetidos en la lista CLOSED y el nodo no se volvería a introducir en la lista OPEN *(no mejora f(n))*.

2. Búsqueda voraz: evaluación

- Completa:
  - Tree Search: NO, se puede quedar estancado en un ciclo.

    Por ejemplo, si se quiere ir de Iasi a Fagaras: Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt (callejón sin salida, bucle infinito)

    Se asemeja a primero en profundidad (prefiere seguir un único camino al objetivo)
  - GRAPH-SEARCH: SI, es completa
- Óptima:
  - No, en cada paso escoge el nodo que se estima más cercano al objetivo (voraz)
- Complejidad temporal:
  - $O(b^m)$ donde m es la profundidad máxima del espacio de búsqueda
  - La utilización de buenas heurísticas puede mejorar la búsqueda notablemente
  - La reducción del espacio de búsqueda dependerá del problema particular y la calidad de la heurística
- Complejidad espacial:
  - $O(b^m)$ donde m es la profundidad máxima del espacio de búsqueda

3. Búsqueda A*

Búsqueda A:
f(n)= g(n) + h(n)

A es el algoritmo más conocido de búsqueda primero el mejor

Combina $g(n)$, el coste de alcanzar el nodo n, y $h(n)$, la estimación heurística del coste a meta:

$$f(n) = g(n) + h(n)$$

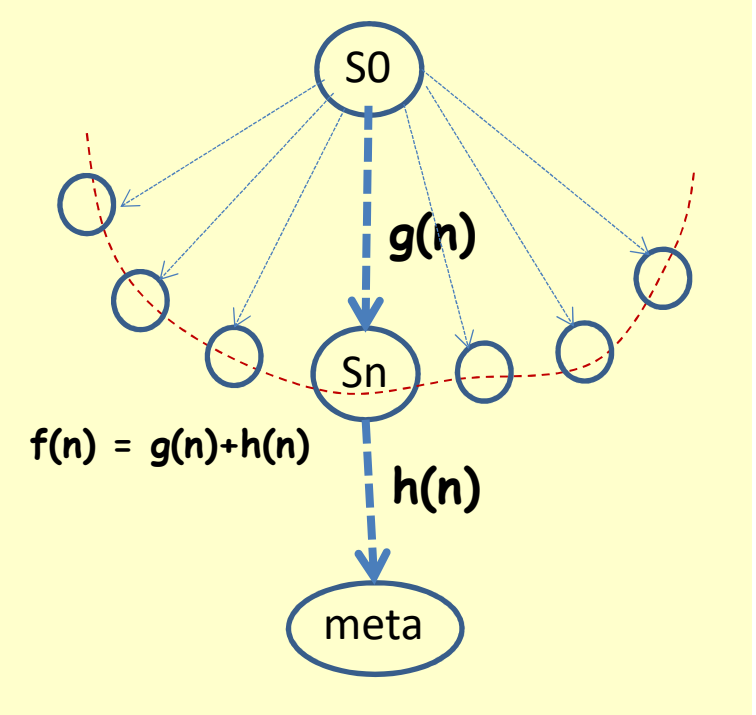
f(n) es el coste total estimado de la solución óptima a través del nodo n

$h(n) >= 0$ y $h(G)=0$ para cualquier objetivo G

Los algoritmos que utilizan una función de evaluación de tipo:

$$f(n) = g(n) + h(n)$$

se denominan algoritmos de tipo A.



3. Búsqueda A*

Búsqueda A*: $f(n)= g(n) + h(n)$

Donde $\forall n$, $\mathbf{h(n) \leq h^*(n)}$

La búsqueda A* utiliza una función heurística admisible

Una heurística es admisible si nunca sobreestima el coste para lograr el objetivo.

Formalmente:

- Una heurística h(n) es admisible si $\forall n$, $\mathbf{h(n) \leq h^*(n)}$, donde $h^*(n)$ es el coste real de alcanzar el objetivo desde el estado **n**.

- Al utilizar un heurístico admisible, la búsqueda A* devuelve la solución óptima

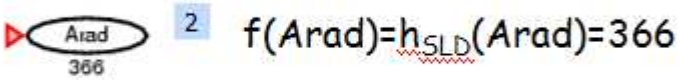
Ejemplo: $h_{SLD}(n)$, *'distancia en línea recta'*, nunca sobreestima la distancia en carretera real entre dos ciudades

3. Búsqueda A*: el ejemplo de Rumanía

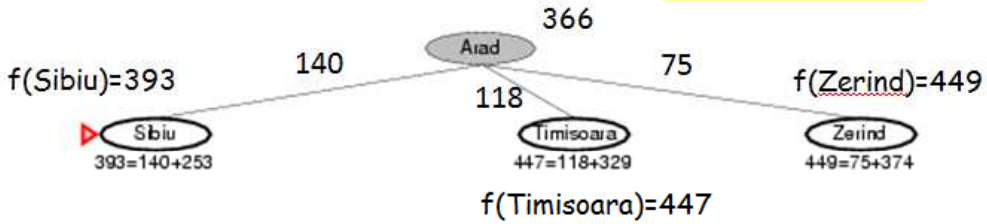
$f(n)=g(n)+h(n)$

- $h(n)=h_{SLD}(n)$
- Expande nodo con el menor coste estimado total f(n)
- Búsqueda A*

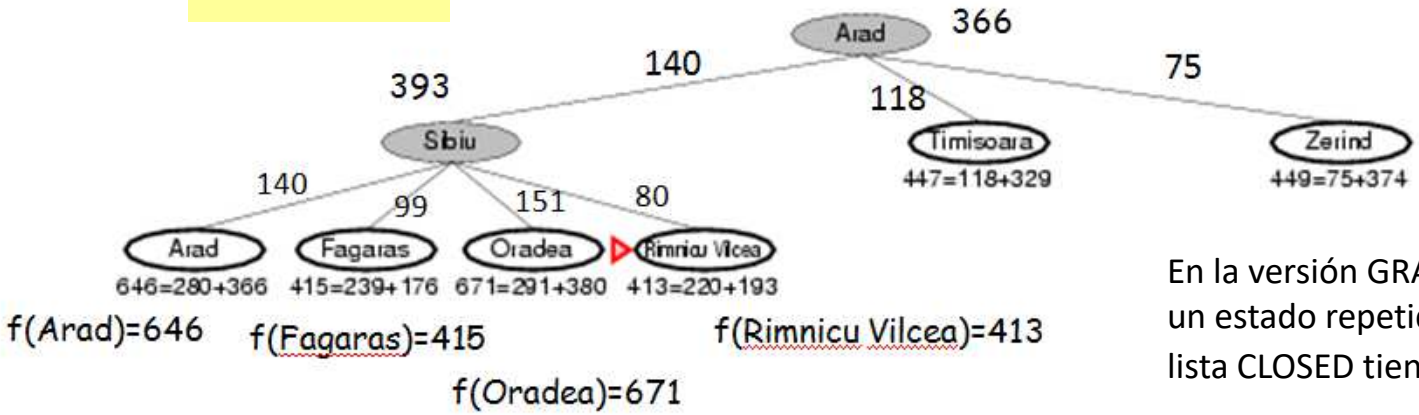
Iteración 1:



Iteración 2:



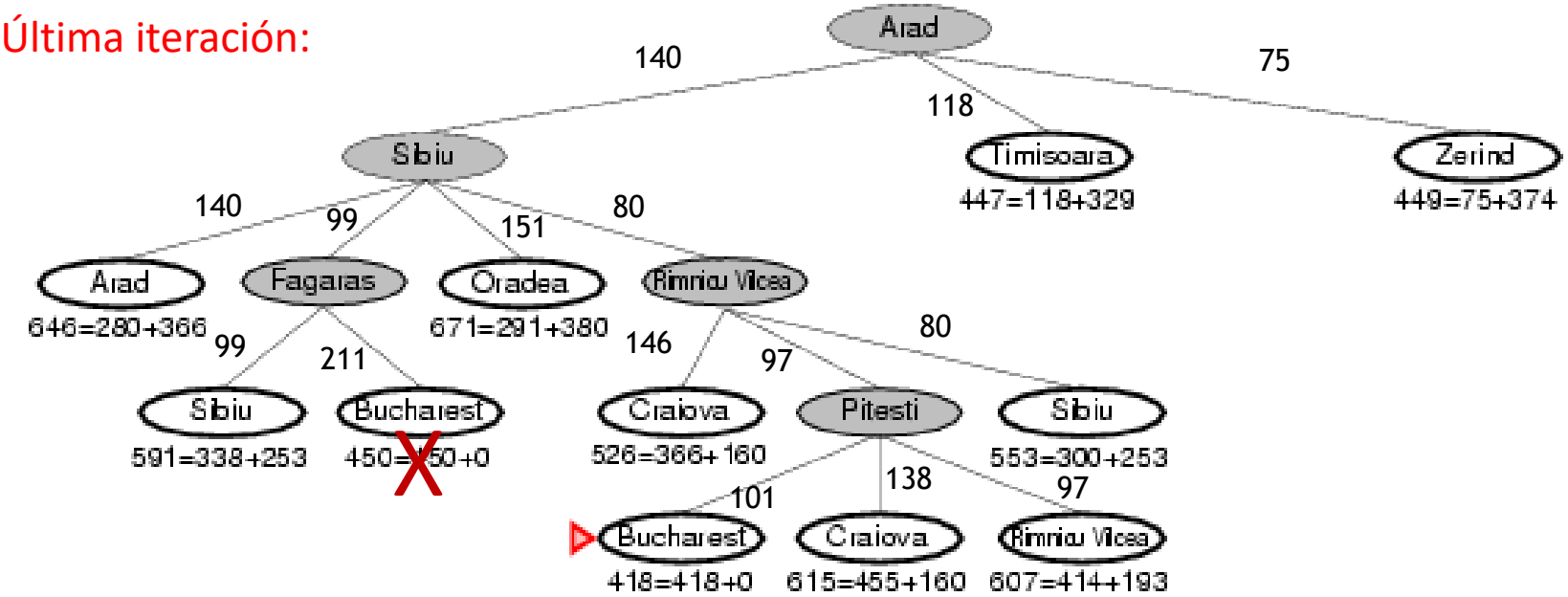
Iteración 3:



En la versión GRAPH-SEARCH: Arad es un estado repetido; el nodo Arad de la lista CLOSED tiene un coste mejor.

3. Búsqueda A*: el ejemplo de Rumanía

Última iteración:



lista OPEN= {Bucarest(418), Timisoara(447), Zerind(449), Craiova(526), Oradea (671)}

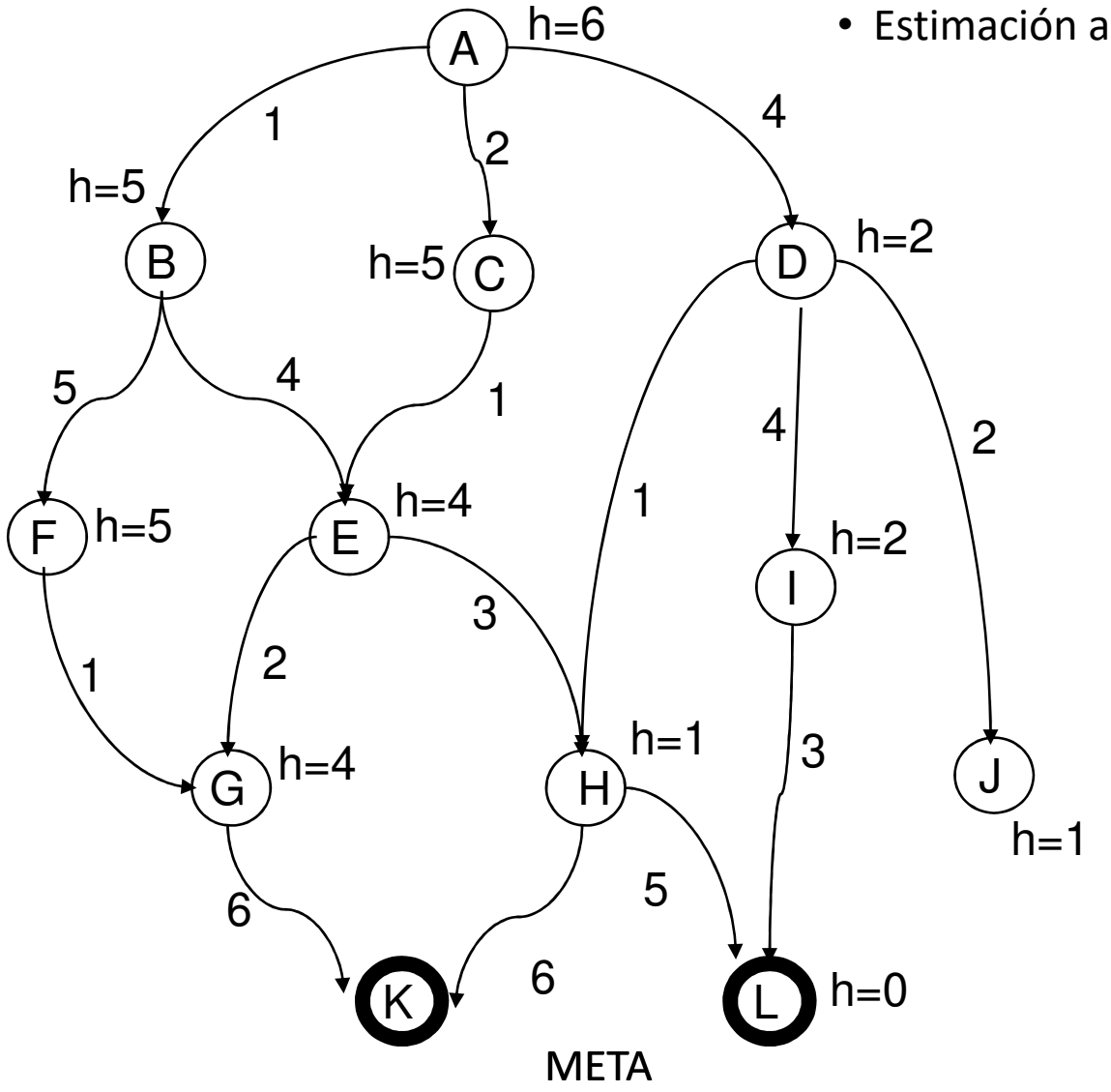
lista CLOSED = {Arad, Sibiu, Rimnicu Vilcea, Fagaras, Pitesti}

El nodo Bucarest ya está en OPEN con coste=450. El nuevo nodo tiene un coste estimado menor (coste=418) que el nodo que está en OPEN. Reemplazamos el nodo de Bucarest en OPEN con el nuevo nodo encontrado.

3. Búsqueda A*: otro ejemplo

Tenemos:

- Coste de acciones: g(n)
- Estimación a meta: h(n)



RECORDATORIO

a) Búsqueda en Árbol: Solo lista OPEN (frontera), ordenados según f(n)

Expansión de un nodo: se elimina de OPEN y sus sucesores se añaden a OPEN.

Control de nodos repetidos en OPEN: Si nuevo nodo mejora nodo repetido en OPEN, nuevo nodo reemplaza al viejo). Si no mejora, se olvida.

• Se pueden generar ciclos y caminos redundantes!.

b) Búsqueda en Grafo. Lista OPEN (frontera) y CLOSED (nodos expandidos).

Expansión de un nodo: se elimina de OPEN y se añade a CLOSED.

a) Nodo generado es nuevo (no en OPEN ni en CLOSED): se añade a OPEN

b) Nodo repetido en OPEN: Reemplaza al viejo si mejora su f(n). Si no, se olvida.

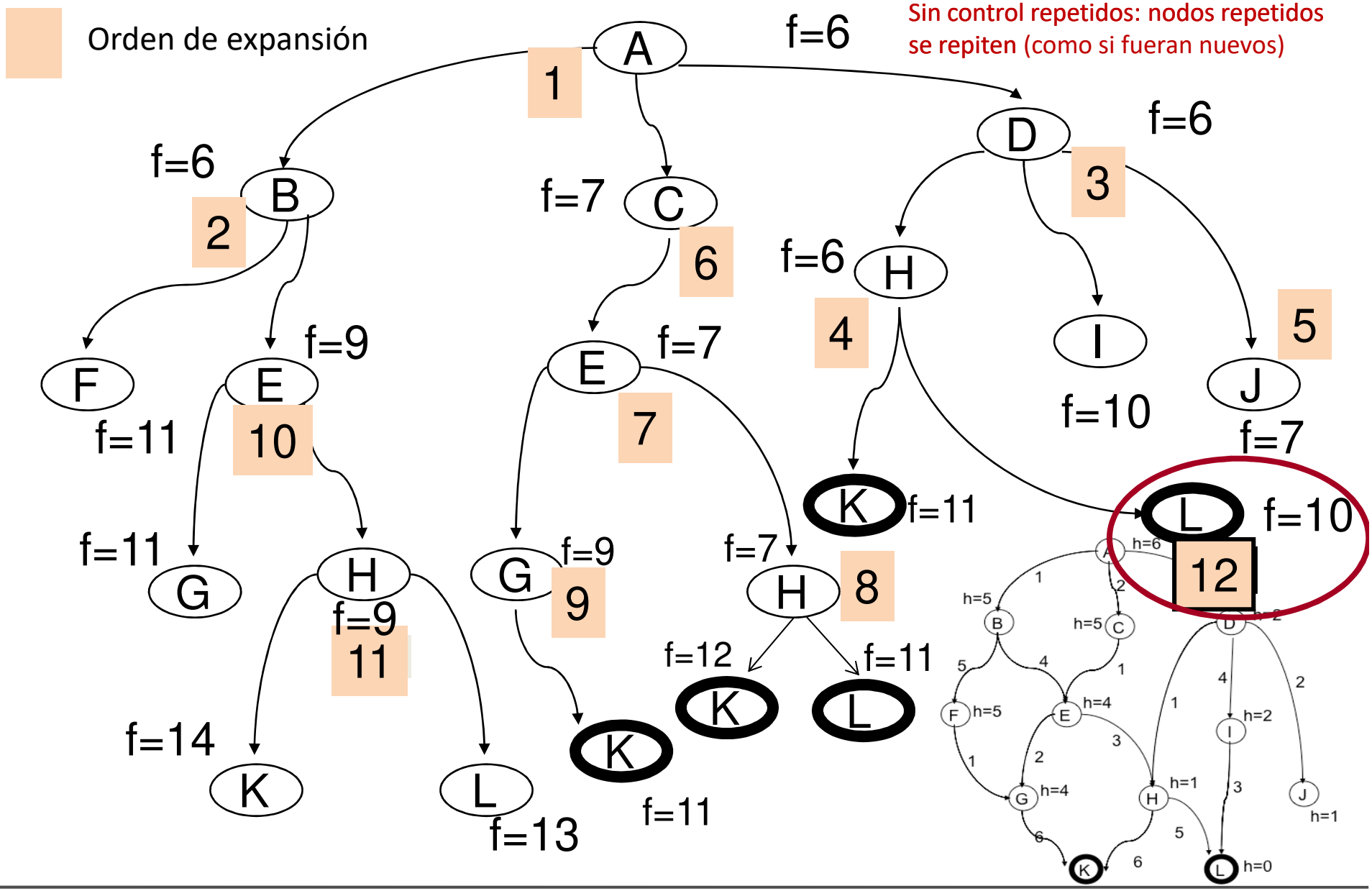
c) Nodo repetido en CLOSED:

Si mejora f(n) re-expandir, se inserta en OPEN (*no ocurrirá si h consistente)*.

Si no mejora, se puede olvidar.

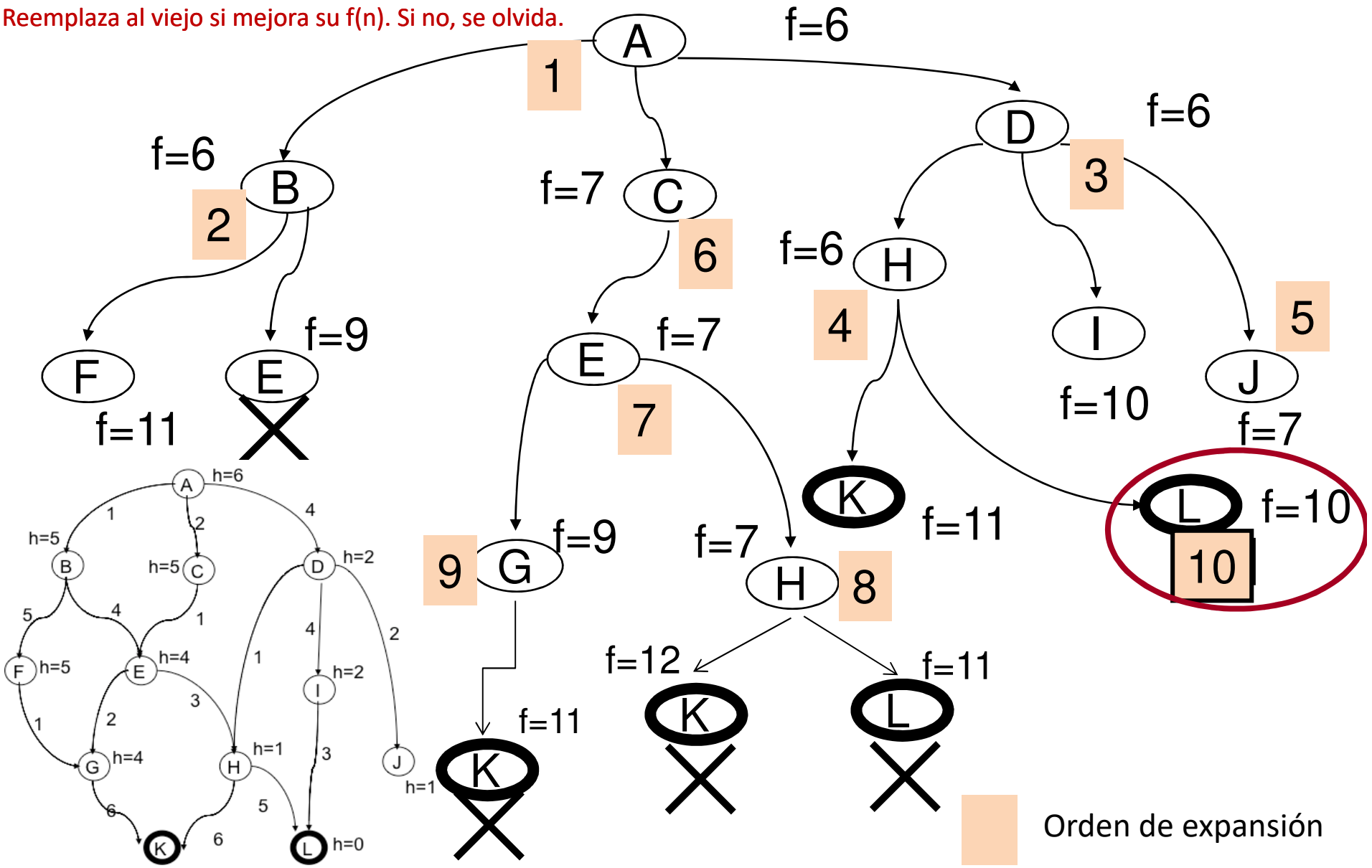
• No se generan caminos redundantes ni ciclos.

3. Búsqueda A*: versión TREE-SEARCH sin control de nodos repetidos

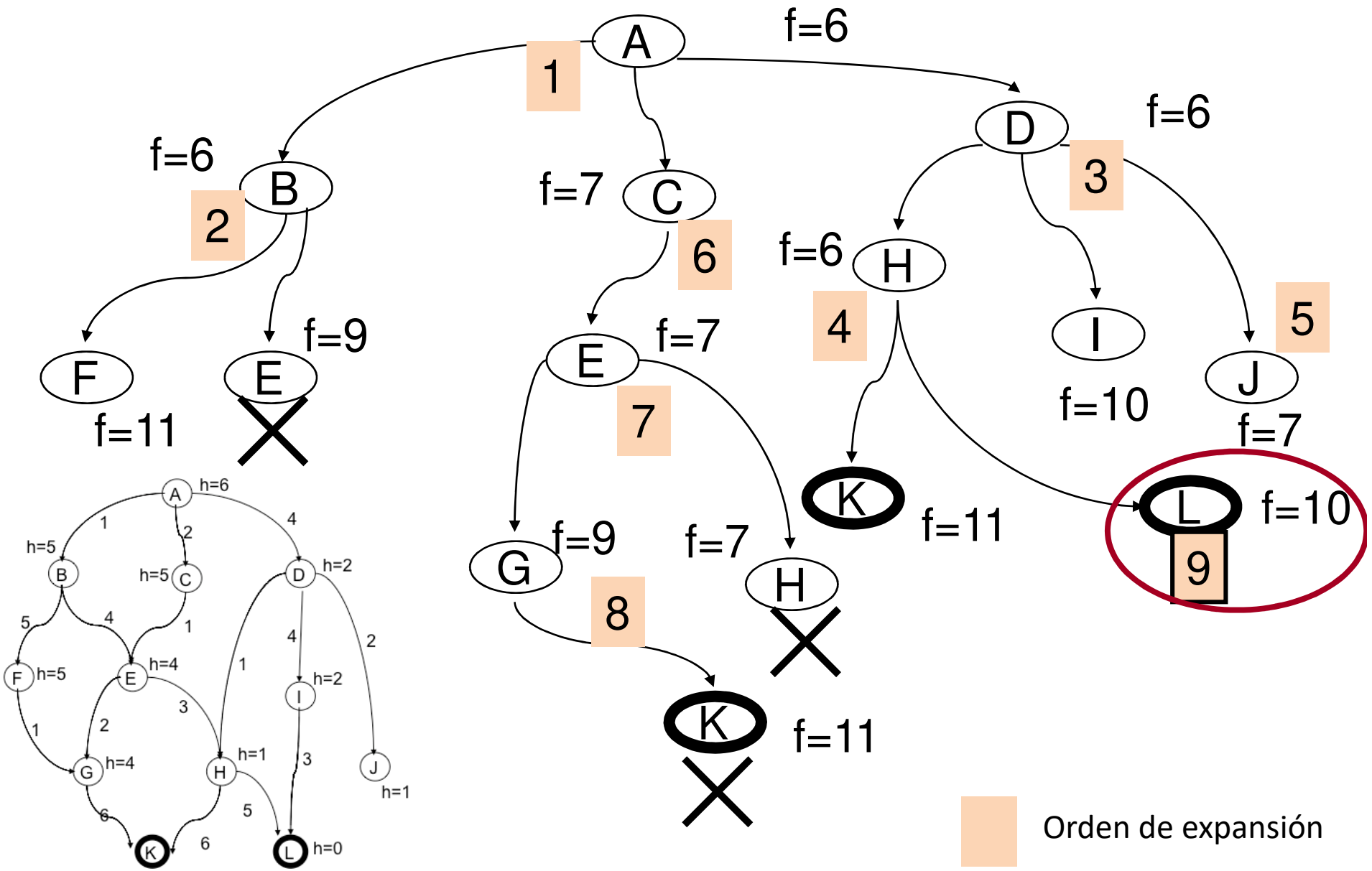


3. Búsqueda A*: versión TREE-SEARCH con control de nodos repetidos

Con control repetidos: Nodo repetido en OPEN:
Reemplaza al viejo si mejora su f(n). Si no, se olvida.



3. Búsqueda A*: versión GRAPH-SEARCH



3. Búsqueda A*: comparación y análisis

- Comparación con otras estrategias de búsqueda:
  - Búsqueda en anchura (óptima si todos los operadores tienen el mismo coste). Equivalente a f(n)=nivel(n)+0, donde $h(n)=0 < h^*(n)$
  - Coste uniforme (óptima). Equivalente a f(n)=g(n)+0 donde $h(n)=0 < h^*(n)$
  - Profundidad (no óptima). No comparable a A*

- Conocimiento heurístico:
  - $h(n)=0$, ausencia de conocimiento
  - $h(n)=h^*(n)$, conocimiento máximo
  - Si $h2(n) \geq h1(n)\ \forall n$ (ambos admisible) entonces h2 domina a h1:
    h2 es más informado que h1; h2 nunca expandirá más nodos que h1

3. Búsqueda A*: comparación y análisis

h(n) = 0: coste computacional de h(n) nulo. (Búsqueda lenta. Admisible.)

h(n)=h*(n): coste computacional grande de h(n). (Búsqueda rápida. Admisible).

h(n)> h*(n): coste computacional de h(n) muy alto. (Búsqueda muy rápida. No admisible)

En general, h* no es conocido pero es posible establecer si h es una cota inferior de h* o no.

Para problemas muy complejos se recomienda reducir el espacio de búsqueda. En estos casos, merece la pena utilizar h(n) > h*(n) con el objetivo de encontrar una solución en un coste razonable incluso aunque ésta no sea la solución óptima.

Para cada problema, encontrar un equilibrio entre el coste de búsqueda y el coste del camino solución.

3. Búsqueda A*: optimalidad

Garantía de Optimalidad:
- ***Tree-Search: Admisible***
- ***Graph-Search: Admisible & Consistente***

La optimalidad de un algoritmo A* TREE–SEARCH se garantiza si h(n) es una heurística admisible

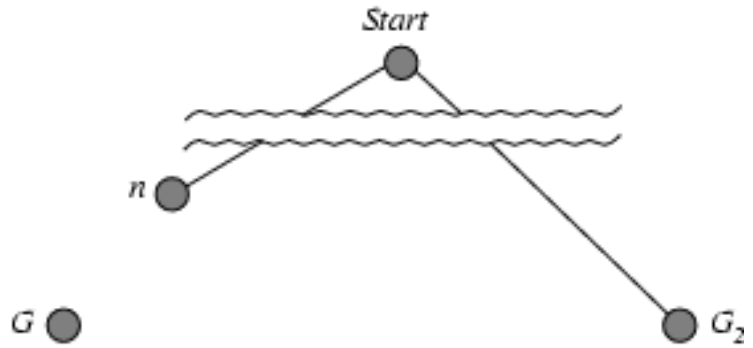
(independientemente de uqe tenga o no control nodos repetidos en OPEN)

Admisibilidad:

- h(n) es admisible si $\forall n,\ h(n) \leq h^*(n)$
- Sea G un estado objetivo óptimo, y n un nodo en el camino a G . Entonces, f(n) nunca sobreestima el coste del camino a través de n.

$$f(n) = g(n) + h(n) \leq g(n) + h^*(n) = g(G) = f(G) \Rightarrow f(n) \leq g(G) \quad (1)$$

Probar que si h(n) es admisible, entonces A* es óptimo:



G es un estado objetivo óptimo: $f(G) = g(G)$

$G2$ es un objetivo subóptimo en la lista OPEN con coste $f(G2) = g(G2) > g(G)$ (2)

n es un nodo de la lista OPEN en el camino óptimo a G.

Si no se escoge n para expansión entonces $f(n) >= f(G2)$ (3)

Si combinamos (1) y (3) :

$$f(G2) <= f(n) <= g(G) \Rightarrow g(G2) \leq g(G)$$

Esto contradice (2) así que se escoge n

Si n en camino de un nodo meta óptimo (G), siempre se expande antes que un nodo meta sub-óptimo (G2)
$f(n) \leq f(G) <= f(G2)$, siempre expande n antes que G2

3. Búsqueda A*: optimalidad

Garantía de Optimalidad:
- ***Tree-Search: Admisible***
- ***Graph-Search: Admisible & Consistente***

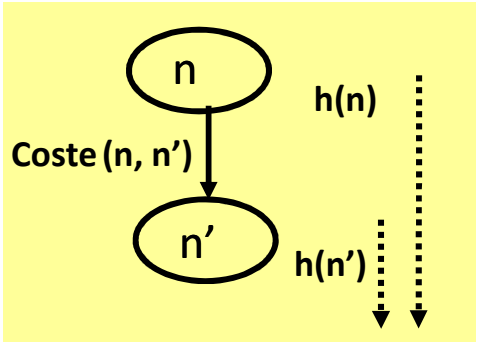
Algoritmo A* GRAPH–SEARCH con re-expansión: Garantiza que se encuentra la solución óptima si h(n) es una heurística admisible.

Algoritmo A* GRAPH–SEARCH sin re-expansión: Garantiza que se encuentra la solución óptima si se puede asegurar que el primer camino que se encuentra a un nodo es el mejor camino hasta dicho nodo (heurística consistente).

Consistencia: h(n) es consistente si, para cada nodo **n** y cada sucesor **n'** de **n** se cumple

$$h(n) <= h(n') + c(n, n')$$

O bien: $h(n') >= h(n) - c(n,n')$, $f(n') >= f(n)$



Consistencia (también llamada *monotonicidad*) es una *condición ligeramente más fuerte* que la admisibilidad.

En resumen,

El algoritmo GRAPH-SEARCH sin re-expansión con h(n) consistente devuelve la solución óptima:

- la secuencia de nodos expandidos por GRAPH-SEARCH es una función no decreciente de f(n).
- Se garantiza que cuando se expande un nodo n, YA se ha encontrado la senda óptima a dicho nodo, por lo que no es necesario volver a re-expandir en caso de generar un nodo repetido (n').

  El coste g(n') del nuevo nodo (n') repetido nunca será mejor que el coste del nodo ya expandido.

3. Búsqueda A*: evaluación

- Completo:
  - Sí, a menos que existan infinitos nodos **n** tal que $f(n) < f(G)$
- Óptima:
  - Sí, si se cumple la condición de:

    Consistencia (para la versión GRAPH-SEARCH), o Admisibilidad para TREE-SEARCH.

    La consistencia requiere admisibilidad (es más fuerte).
  - Siempre existe al menos un nodo **n** en OPEN que pertenece al camino óptimo de la solución
  - Si C^* es el coste de la solución óptima:
    - A* expande todos los nodos con $f(n) < C^*$
    - A* podría expandir algunos nodos con $f(n)=C^*$.
    - A* no expande nodos con $f(n) > C^*$

- Complejidad temporal:
  - $O(b^{C^*/\text{min\_coste\_acción}})$: exponencial con la longitud de camino *(similar coste uniforme)*
  - El número de nodos expandidos es exponencial con la longitud de la solución

- Complejidad espacial:
  - Mantiene todos los nodos en memoria (como todas las versiones GRAPH-SEARCH)
  - A* normalmente se queda sin espacio mucho antes de que se agote el tiempo
  - Por tanto, el mayor problema es el espacio, no el tiempo

Búsqueda en IA - Obtener senda de mínimo coste desde raíz a un estado meta.

a) Búsqueda en Árbol: Solo lista OPEN (frontera), ordenados según f(n)
Expansión de un nodo: se elimina de OPEN y sucesores se añaden a OPEN.
Control repetidos: Nodo repetido reemplaza al viejo si mejora su f(n). Si no, se olvida.

- Se pueden generar ciclos y caminos redundantes!.

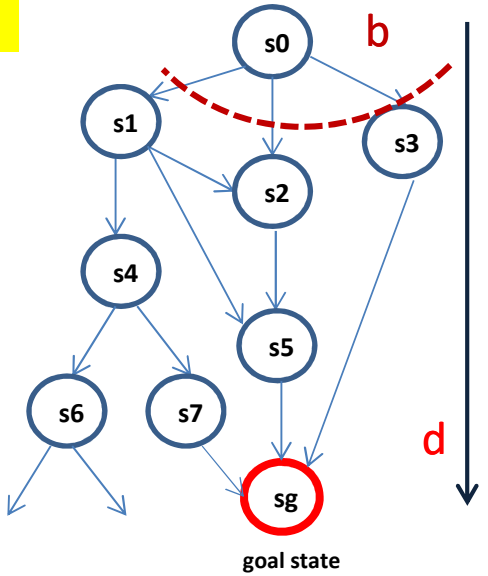
b) Búsqueda en Grafo. Lista OPEN (frontera) y CLOSED (nodos expandidos).
Expansión de un nodo: se elimina de OPEN y se añade a CLOSED.

a) Nodo generado es nuevo: se añade a OPEN

b) Nodo repetido en OPEN: Reemplaza al viejo si mejora su f(n). Si no, se olvida.

c) Nodo repetido en CLOSED: Si mejora f(n) se re-expande: Se inserta en OPEN *(no necesario si h consistente)* Si no, se olvida.

- No se generan caminos redundantes ni ciclos.



No Informada	ANCHURA	C. UNIFOR.	PROFUND	PROF. ITER.
Completa / Óptima	SI/SI*	SI/SI	NO/NO	SI/SI*
C. ESPACIAL	$O(b^d)$	$\approx O(b^d)$	O(b m)	O(b d)
C. TEMPORAL	$O(b^d)$	$\approx O(b^d)$	$O(b^m)$	$O(b^d)$

Informada	VORAZ	Algoritmo A	Algoritmo A*
	f(n) = h(n)	f(n) = g(n) + h(n)	$\forall n, h(n) \leq h^*(n)$
Completa / Óptima	NO	Árbol (o Grafo con re-expansión): Admisibilidad Grafo sin re-expansión: Consistencia h(n)	

- **g(n): coste desde raíz**
- **h(n): Estimación coste a meta**
- **Estrategia de Búsqueda:** Ordenación en OPEN según f(n)
- **Prueba de objetivo:** Cuando el nodo se selecciona para expandirse.

4. Diseño de funciones heurísticas

Las heurísticas son funciones dependientes del problema.

¿Cómo diseñar una función heurística (admisible) para un problema?

Técnica común:

Relajación de las restricciones del problema

Considerar el problema con menos restricciones, obteniendo así otro problema que se resuelve con una complejidad menor que la del problema inicial.

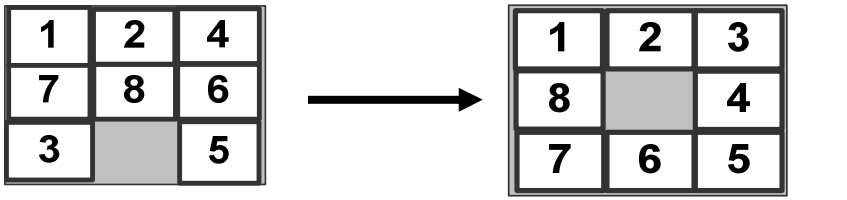
El coste de la solución del problema relajado se utiliza como una estimación (admisible) del coste del problema original.

4.1. Heurísticas para el problema de 8-puzzle

8-puzzle: una ficha situada en una casilla A se puede mover a una casilla B si

Restricción 1: B es adyacente a A

Restricción 2: B es el espacio vacío



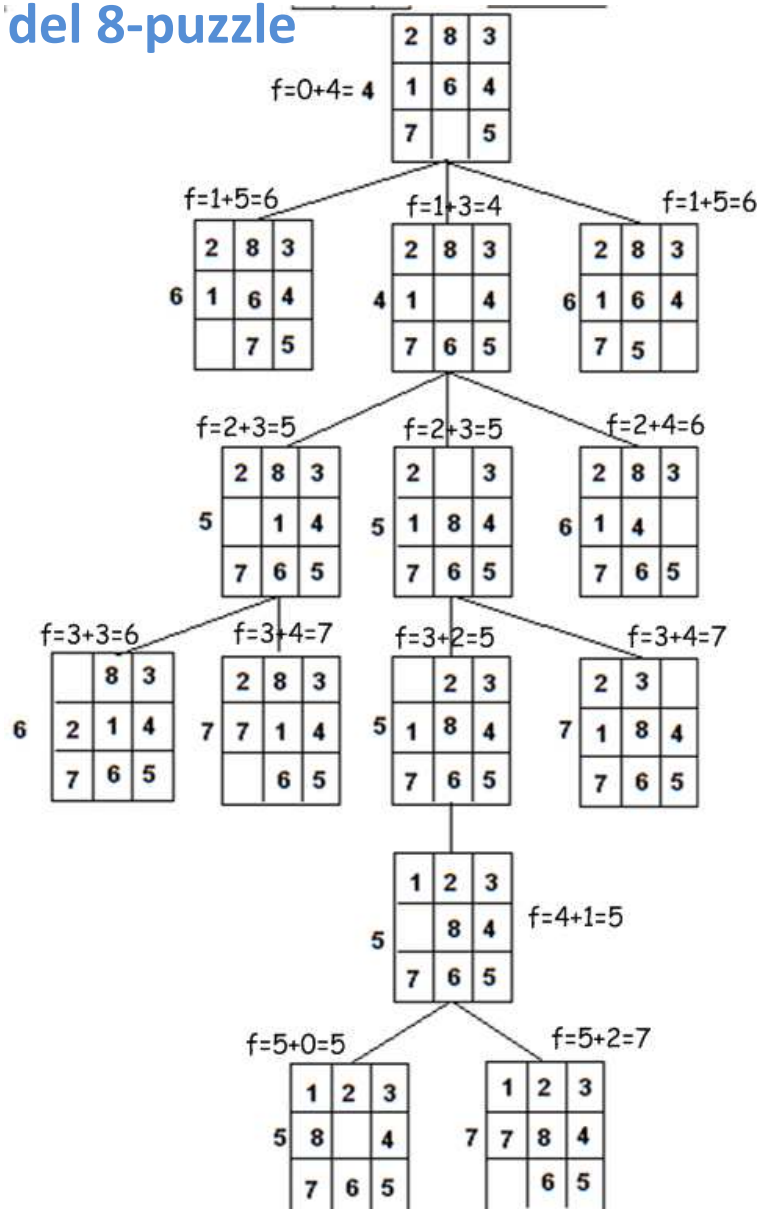
Estado inicial Estado objetivo

- El problema de 8-puzzle:
  - El coste medio de una solución es aproximadamente 22 pasos (factor de ramificación ≈ 3)
  - Búsqueda exhaustiva a profundidad 22: $3^{22} \approx 3.1 \times 10^{10}$ estados
  - Una buena función heurística puede reducir el proceso de búsqueda
- **H1: fichas descolocadas**
  - ✓ h1(n): número de fichas descolocadas: Elimina ambas restricciones
  - ✓ Una ficha se puede mover a cualquier casilla
  - ✓ h1(n) devuelve una estimación muy optimista (solución óptima para el problema relajado)
  - ✓ h1(n)=5 para el ejemplo del estado inicial
- **H2: distancias de Manhattan**
  - ✓ h2(n): suma de las distancias de cada ficha a su posición objetivo
  - ✓ Elimina Restricción 2: Una ficha se puede mover a cualquier casilla adyacente
  - ✓ h2(n) devuelve una estimación optimista (solución óptima para el problema relajado)
  - ✓ h2(n)=1+2+4+1+1=9 para el ejemplo del estado inicial

Distancias de Manhattan domina a fichas descolocadas: $h2(n) >= h1(n), \forall n$

4.1. Heurísticas para el problema del 8-puzzle

Búsqueda A* con la heurística
fichas descolocadas



4.2. Heurísticas para el problema del viajante de comercio

6 ciudades: A,B,C,D,E,F

Estados: secuencias de ciudades que comienzan en la ciudad A y representan rutas parciales

Inicio: A

Final: secuencias que empiezan y terminan en A y pasan por todas las ciudades

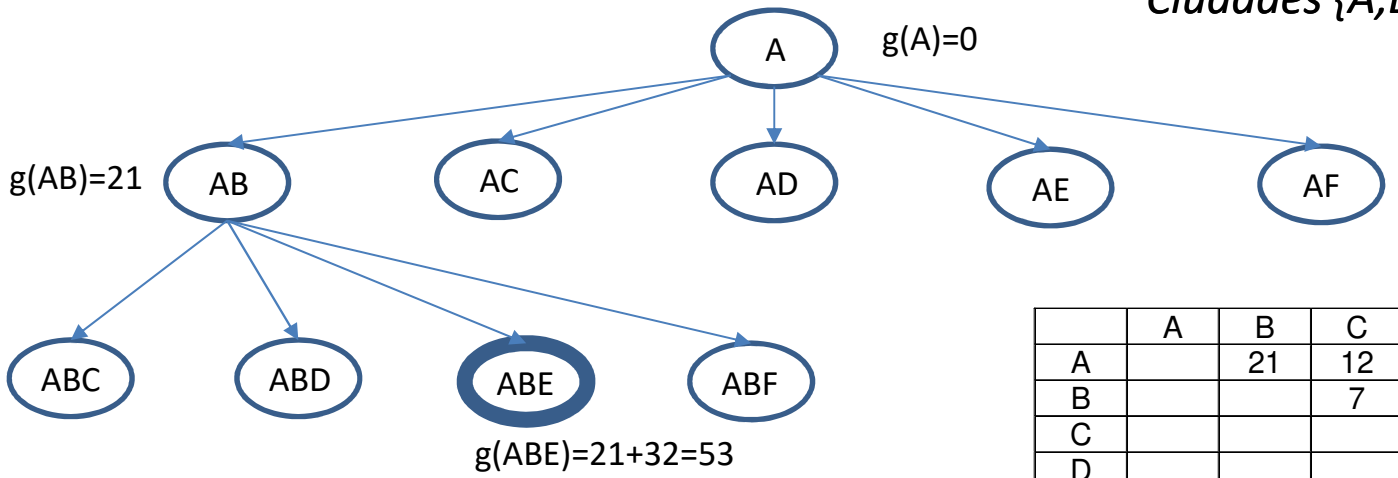
Reglas u operadores: añadir al final de cada estado una ciudad que no está en la secuencia

Coste de los operadores: distancia entre la última ciudad de la secuencia y la nueva ciudad añadida (ver tabla)

	A	B	C	D	E	F
A		21	12	15	113	92
B			7	25	32	9
C				5	18	20
D					180	39
E						17

4.2. Heurísticas para el problema del viajante de comercio

Ciudades {A,B,C,D,E,F}



	A	B	C	D	E	F
A		21	12	15	113	92
B			7	25	32	9
C				5	18	20
D					180	39
E						17

Recorrido Actual: A > B > E

Faltan por visitar: C, D, F, volver a A

¿h(ABE)?

h*(ABE) = min(ECDFA, ECFDA, EDCFA, EDFCA, EFCDA, EFDCA)

min(154, 92, 297, 251, 57, 73)

4.2. Heurísticas para el problema del viajante de comercio

h(ABE): E → {C, D, F}, volver a A

h1(n)=0 en todos los casos (coste uniforme)

h2(n)= número de arcos que faltan multiplicado por el coste del arco mínimo

[h2(ABE)=4*5=20]

h3(n)= suma de los *p* arcos más cortos si faltan *p* arcos (arcos no dirigidos)

[h3(ABE)=5+5+7+7=24]

	A	B	C	D	E	F
A		21	12	15	113	92
B			7	25	32	9
C				5	18	20
D					180	39
E						17

4.2. Heurísticas para el problema del viajante de comercio

h(ABE): E → {C, D, F}, volver a A

h4(n)= suma de los arcos más cortos que parten de las ciudades que faltan por abandonar

[h4(ABE)=17 {abandonar E} +
5 {abandonar C} +
5 {abandonar D} +
9 {abandonar F} =36]

h7(n)= número de arcos que faltan multiplicado por el coste medio de los arcos *(no admisible)*

[h7(ABE)=4*40.33=161.33]

	A	B	C	D	E	F
A		21	12	15	113	92
B			7	25	32	9
C				5	18	20
D					180	39
E						17

h5(n)= suma de los arcos más cortos que salen de las ciudades que quedan por alcanzar
[h5(ABE)= 5 {alcanzar C}+5 {alcanzar D}+9 {alcanzar F} + 12 {alcanzar A } =31]

h6(n)= distancia más corta entre la última ciudad de 'n' y la ciudad de partida

h8(n)= suma de los *p* arcos más grandes si faltan *p* arcos

5. Evaluación de funciones heurísticas

Dificultad de aplicar un análisis matemático

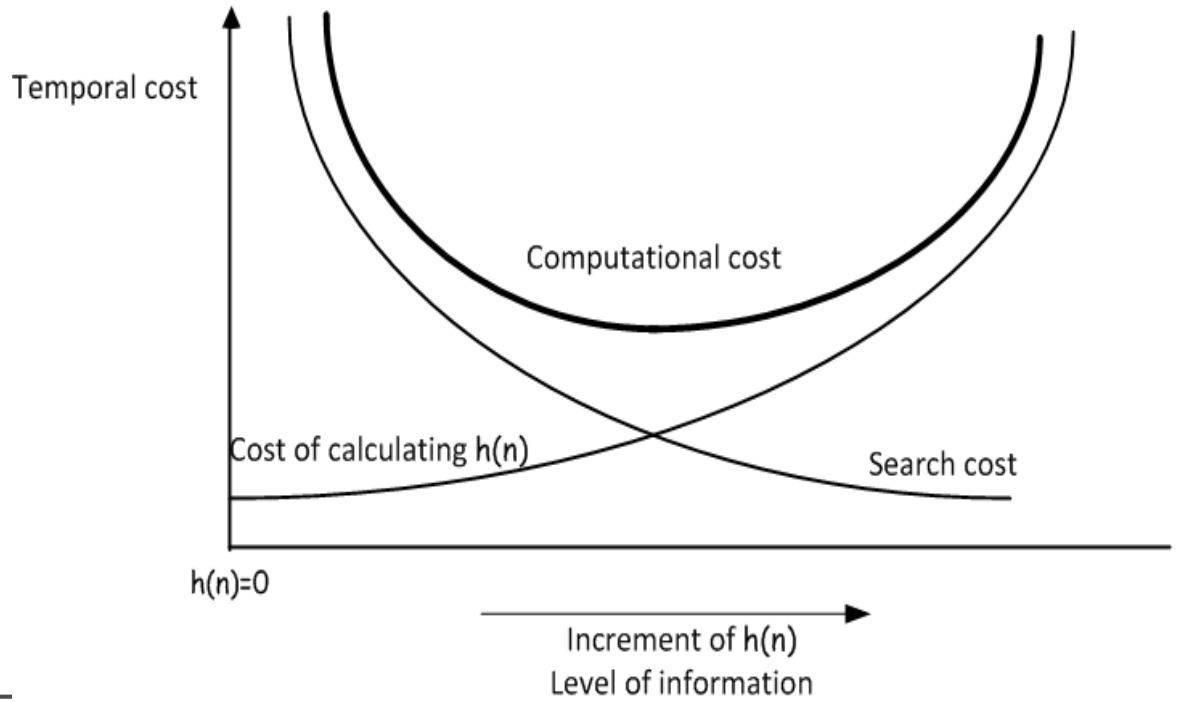
Se aplican métodos estadísticos/experimentales.

Objetivo: buscar balance entre coste de la búsqueda y coste de la solución

Coste computacional (temporal coste):

Coste de búsqueda: número de nodos generados o operadores aplicables **+**

Coste de calcular h(n): coste para seleccionar el nodo (operador aplicable)



5. Evaluación de funciones heurísticas

Factor efectivo de ramaje (b*)

N = Número total de nodos generados por un método de búsqueda para un problema particular

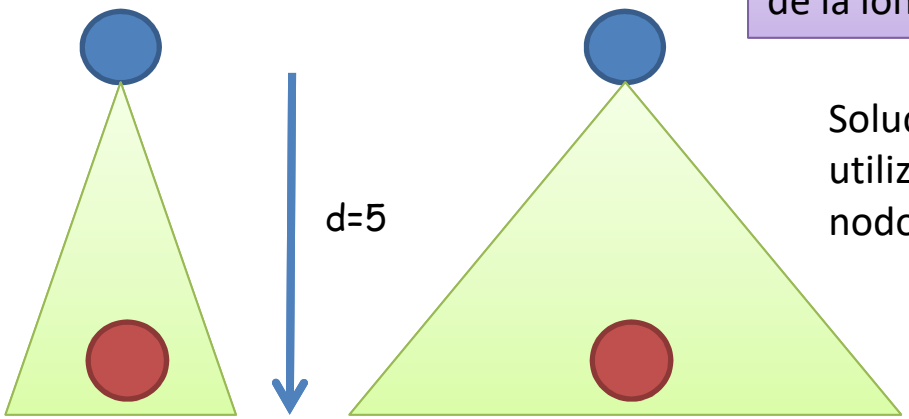
d = Profundidad de la solución

b* = Factor de ramificación de un árbol uniforme de profundidad d con N+1 nodos.

$$N+1=1+b^*+(b^*)^2+ \ldots + (b^*)^d$$

- b* define si la búsqueda hacia el objetivo está bien enfocada o no
- b* es razonablemente independiente de la longitud del camino (d)

Solución en d=5 utilizando N=52 nodos → b*=1.92



Solución en d=5 utilizando N=560 nodos → b*=3.3

Una heurística bien diseñada tendría un valor de b* cercano a 1, lo que permitiría resolver bastantes instancias complejas del problema.

Un valor de b* cercano a 1 corresponde a una búsqueda que está altamente enfocada hacia la meta, con muy poca ramificación en otras direcciones.

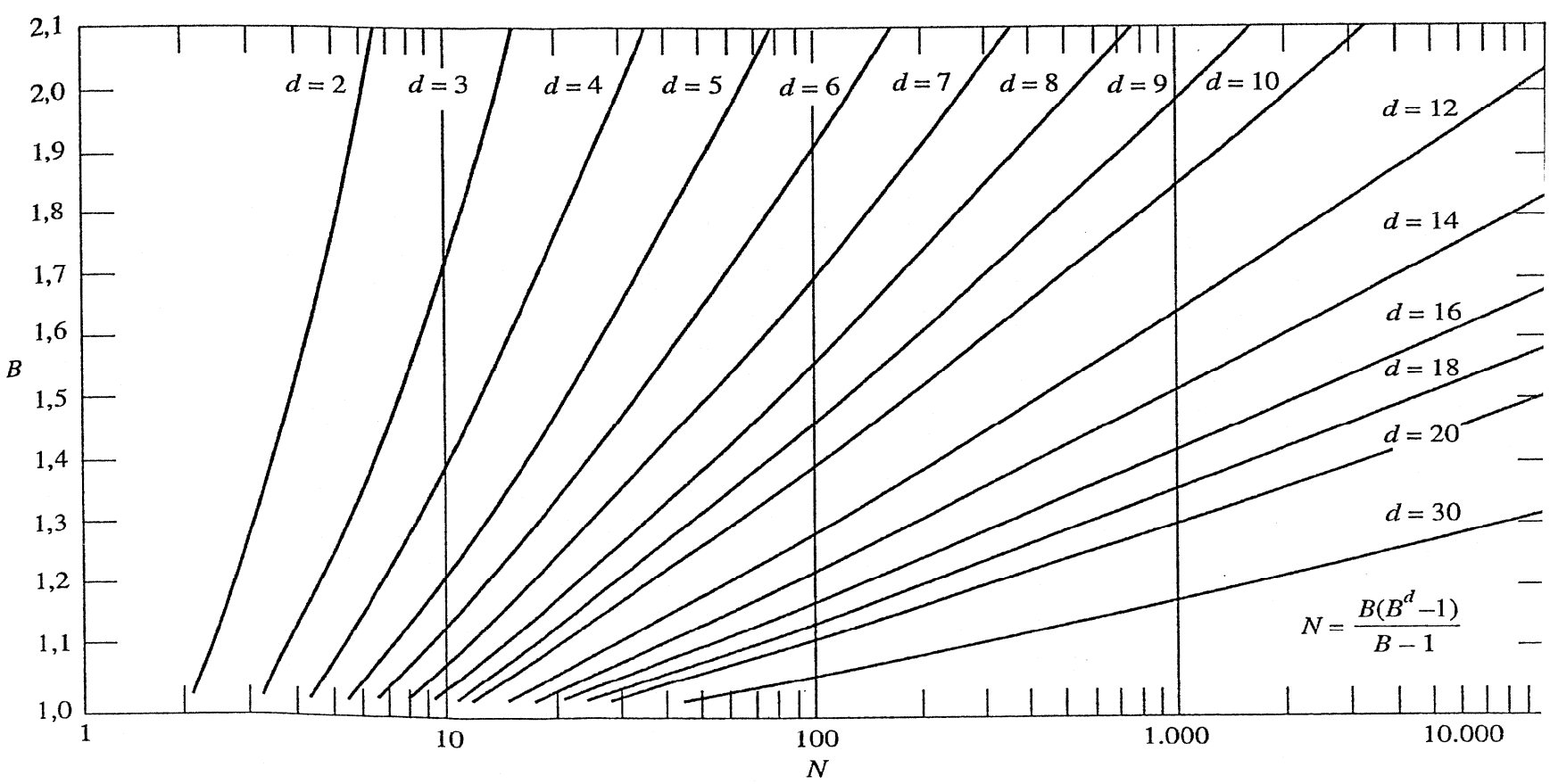


Figura 9.11.

B frente a *N* para distintos valores de *d*.

Bloque 1 – Representación de Conocimiento y Búsqueda

Tema 5:
Búsqueda Heurística

RESUMEN

Búsqueda en IA

Objetivo: Obtener senda de mínimo coste desde raíz a un estado meta.

a) **Búsqueda en Árbol**: Solo lista OPEN (frontera), ordenados según f(n)
Expansión de un nodo, se elimina de OPEN y sucesores se añaden a OPEN.
   a) Sin control repetidos: nodos repetidos se repiten (como si fueran nuevos)
   b) Con control repetidos: Nodo repetido en OPEN: Reemplaza al viejo si mejora su f(n).

- Se pueden generar ciclos y caminos redundantes.

b) Búsqueda en Grafo. Lista OPEN (frontera) y CLOSED (nodos expandidos).

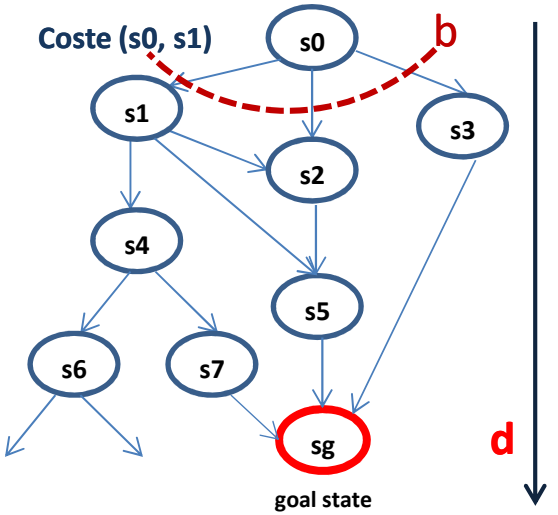
- Cuando se expande un nodo, se elimina de OPEN y se añade a CLOSED.

a) Nodo generado es nuevo (no en OPEN ni en CLOSED): se añade a OPEN

b) Nodo repetido en OPEN: Reemplaza al viejo si mejora su f(n). Si no, se olvida.

c) Nodo repetido en CLOSED: Si mejora f(n)se re-expande: Se inserta en OPEN. (*No necesario si consistente*). Si no, se olvida

- No se generan caminos redundantes ni ciclos.



- g(n): coste desde raíz
- h(n): estimación a meta
- **Estrategia de Búsqueda:** Ordenación frontera en OPEN según f(n)
- **Prueba de objetivo:** Cuando el nodo se selecciona para expandirse.

	ANCHURA	C. UNIFOR.	PROFUND	PROF. ITER.	Con Informacion h(n) VORAZ	Tipo A
CRITERIO	f(n)=Nivel(n)	f(n)=g(n)	- Nivel	Prof + Anch	f(n)=h(n)	f(n)=g(n)+h(n)
COMPLETA	SI	SI	NO	SI	NO	SI
ÓPTIMA	SI*	SI	NO	SI*	NO	SI, h(n)< h*(n) Si,h(n) consistente
C. ESPACIAL	$O(b^d)$	$\approx O(b^d)$	O(b m)	O(b d)	$O(b^m)$	$\approx O(b^d)$
C. TEMPORAL	$O(b^d)$	$\approx O(b^d)$	$O(b^m)$	$O(b^d)$	$O(b^m)$	$\approx O(b^d)$

Optimalidad Algoritmo A

Algoritmo de Búsqueda: Ordenación de los nodos de la lista OPEN (frontera) para su expansión.

Algoritmo A: los nodos se ordenan en OPEN según $f(n) = g(n) + h(n)$, donde:

g(n): coste (conocido) de nodo inicial a nodo n
h(n): estimación del coste del nodo n a nodo meta.

h(n) es ADMISIBLE si, $\forall n, h(n) \leq h^*(n)$, donde $h^*(n)$ es el coste óptimo de alcanzar el objetivo desde n

Un algoritmo es del **tipo A*** cuando: $\forall n:\{ h(n) \leq h^*(n)\}$. Es decir, h(n) es **admisible**

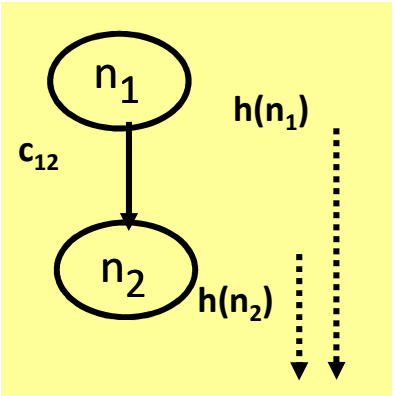
h(n) es CONSISTENTE si, siendo n_2 sucesor de n_1, se cumple que:

$$\forall n \quad h(n_2) \geq h(n_1) - Coste(n_1 \rightarrow n_2)$$

f(n) es monotonamente creciente a medida que se desciende en el árbol *(cumple la **propiedad de monotonía**)*: $\forall n \quad f(n_2) \geq f(n_1)$

Cuando se genere un nuevo nodo, ya expandido (en CLOSED), no mejorará su coste y puede olvidarse.

Una heurística consistente es admisible, pero lo contrario no es cierto.



Garantía de optimalidad:

a) **Búsqueda en Árbol**: Si h(n) es **ADMISIBLE**

b) **Búsqueda en Grafo, con re-expansión:** SI h(n) es **ADMISIBLE**

c) **Búsqueda en Grafo, sin re-expansión:** Si h(n) es **CONSISTENTE los nodos repetidos en CLOSED, pueden olvidarse.** *CONSISTENCIA de h(n) implica ADMISIBILIDAD de h(n).*