Bloque 1 – Representación de Conocimiento y Búsqueda

Tema 5:
Búsqueda Heurística

Bloque 1, Tema 5- Índice

1. Búsqueda heurística
2. Búsqueda voraz
3. Búsqueda A*
4. Diseño de funciones heurísticas
   - 4.1 Heurísticas para el problema del 8-puzzle
   - 4.2 Heurísticas para el problema del viajante de comercio
5. Evaluación de funciones heurísticas.

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/

1. Búsqueda heurística

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

Puede 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 comprensión 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**.

2. Búsqueda *voraz*

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

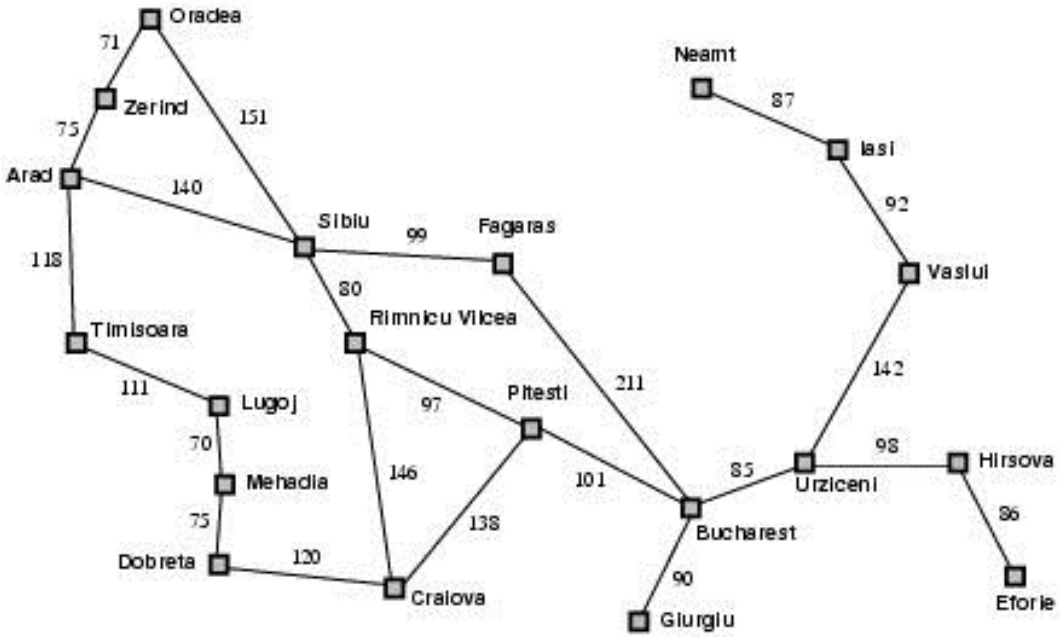
Función heurística: *Simplificación o conjetura que reduce o limita la búsqueda de soluciones en dominios complejos o poco conocidos.*

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. 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

h_{SLD} NO **se puede calcular** a partir de la descripción del problema

Ejemplos:

h(Arad)=366

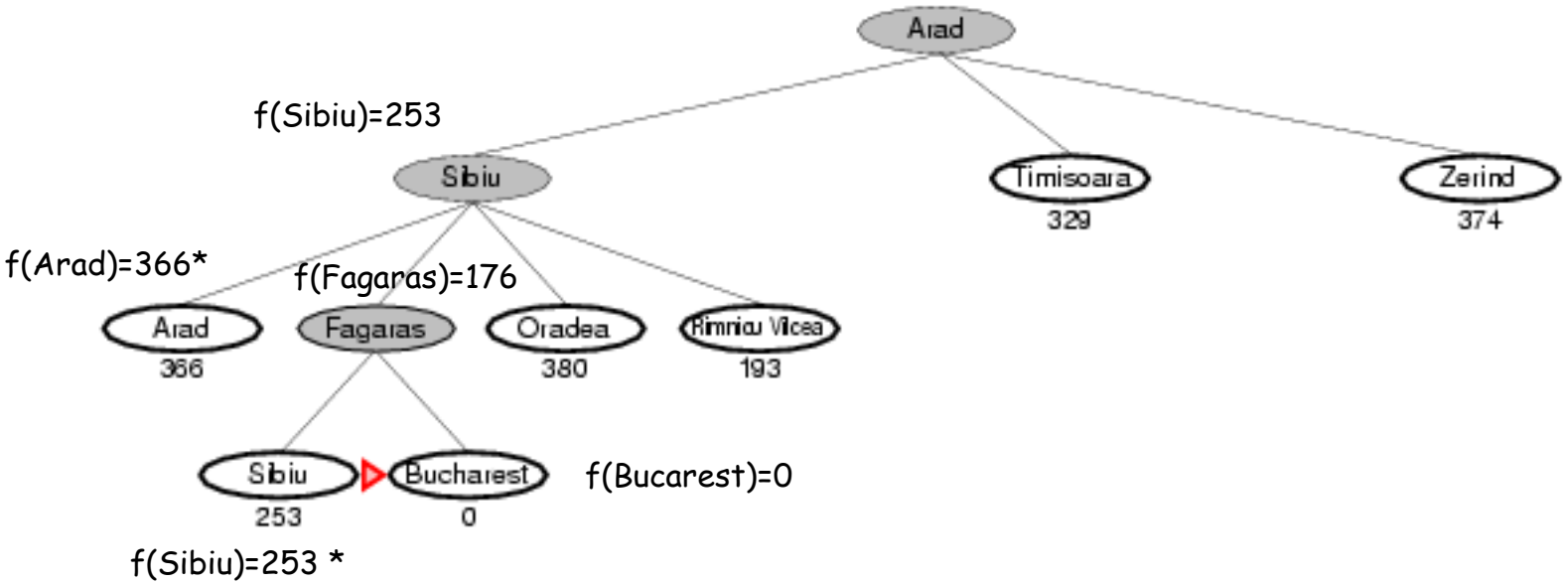
h(Fagaras)=176

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.

2. Búsqueda voraz: evaluación

- Completa:
  - NO, se puede quedar estancado en un ciclo; p. ej. 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)
  - La versión GRAPH-SEARCH es completa
- Óptima:
  - No, en cada paso escoge el nodo 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*

A*es el algoritmo más conocido de búsqueda primero el mejor

Evalúa los nodos combinado *g*(n), el coste de alcanzar el nodo **n**, y h(n), el valor heurístico: **f(n)=g(n)+h(n)**

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

Los algoritmos que utilizan una función de evaluación de la forma **f(n)=g(n)+h(n)** se denominan algoritmos de tipo A.

3. Búsqueda A*

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
- h(n) >= 0 así que h(*G*)=0 para cualquier objetivo *G*

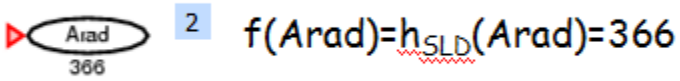
P. ej. $h_{SLD}(n)$ nunca sobreestima la distancia en carretera real entre dos ciudades

3. Búsqueda A*: el ejemplo de Rumanía

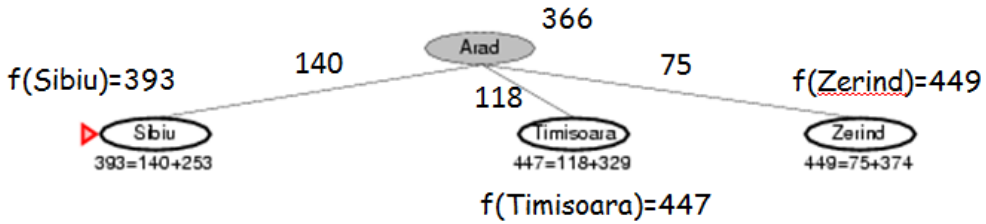
f(n)=g(n)+h(n)

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

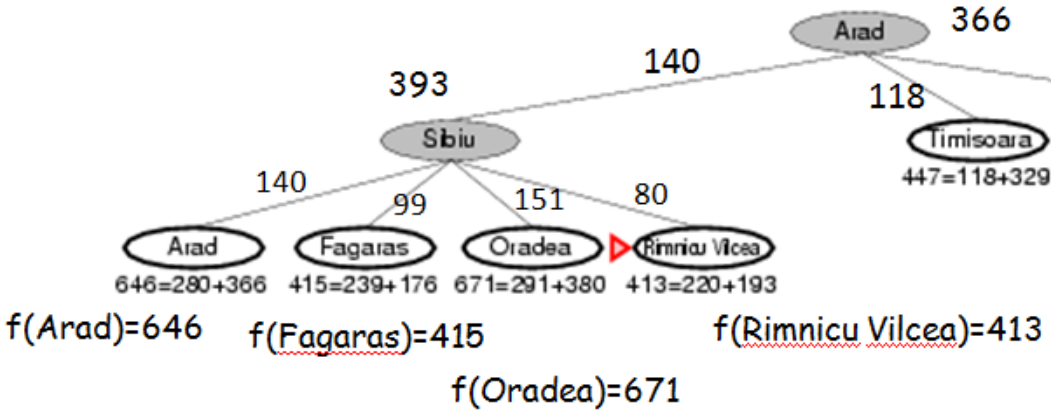
Iteración 1:



Iteración 2:



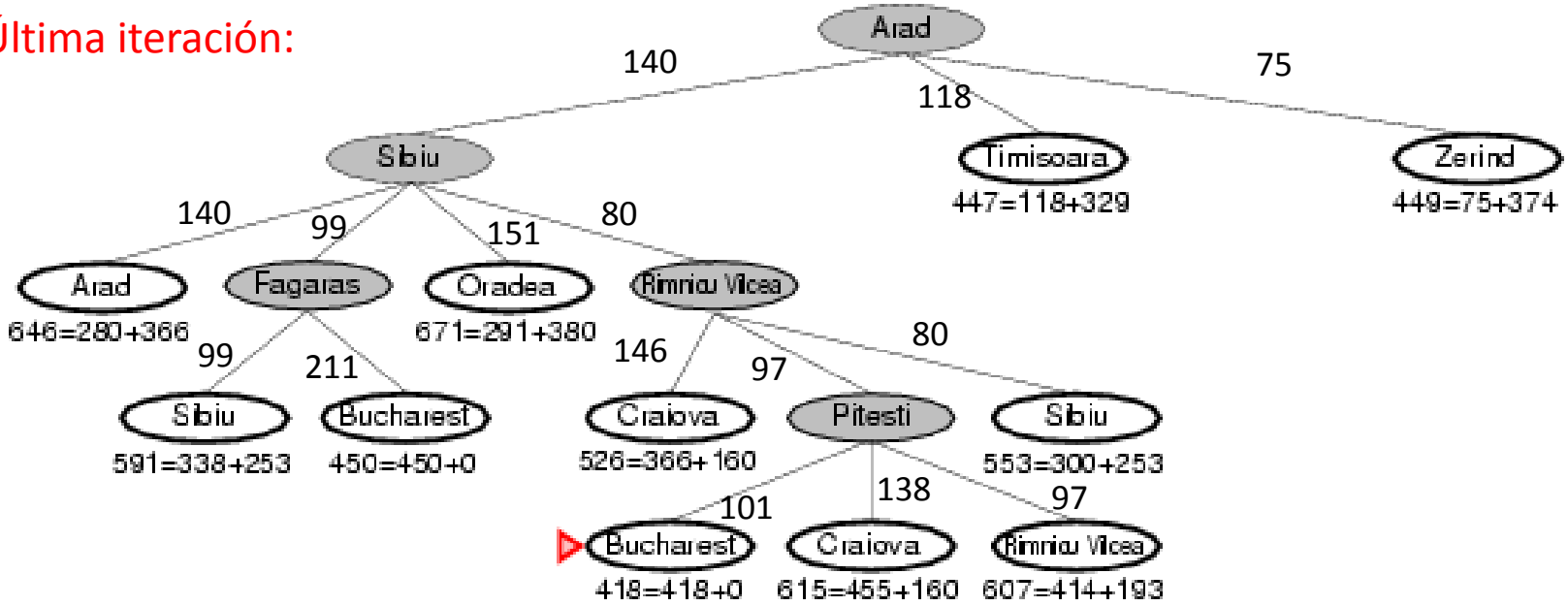
Iteración 3:



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:

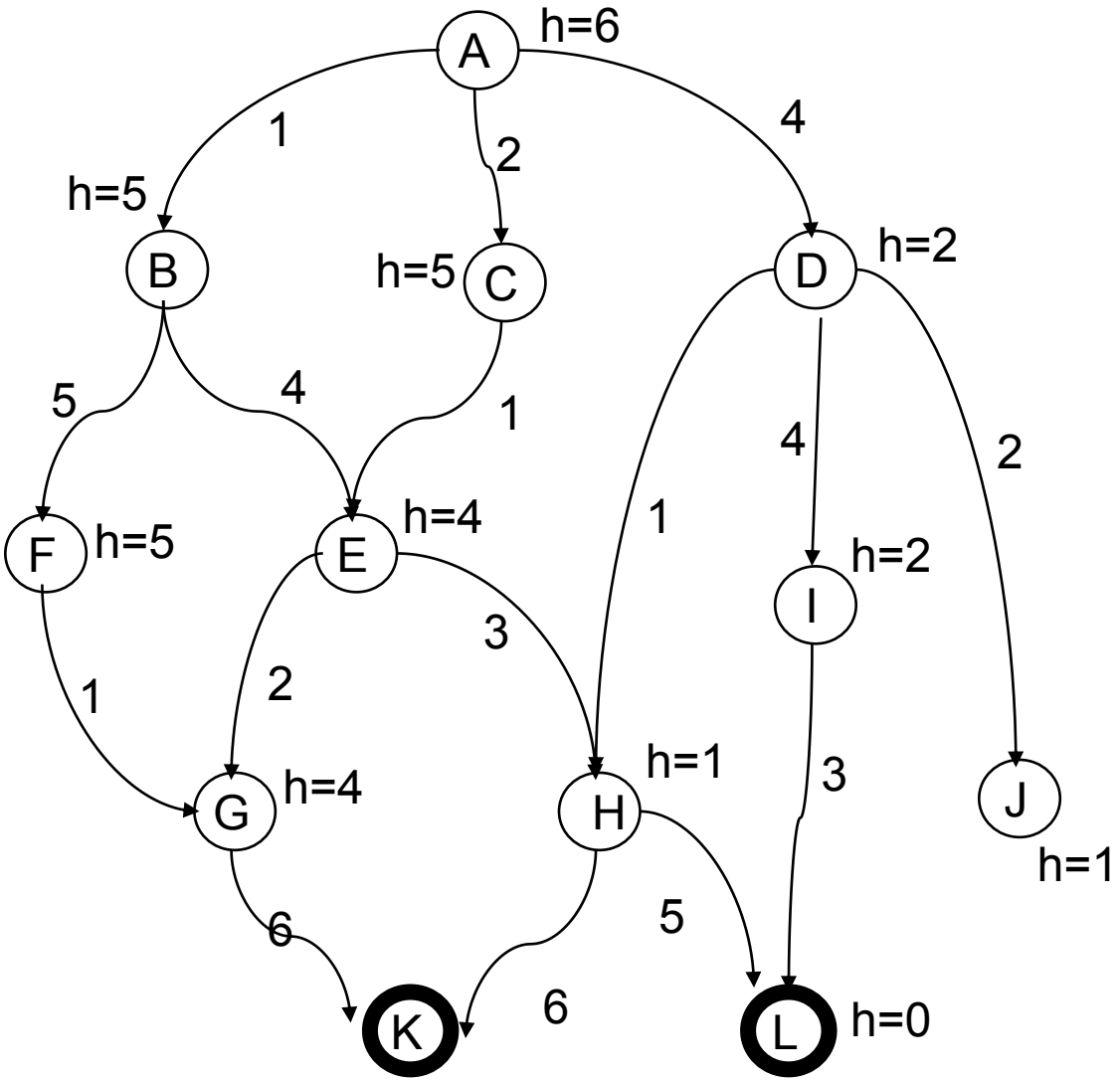


lista OPEN= {Bucarest(418), Timisoara(447), Zerind(449), Craiova(526), Oradea (671)}

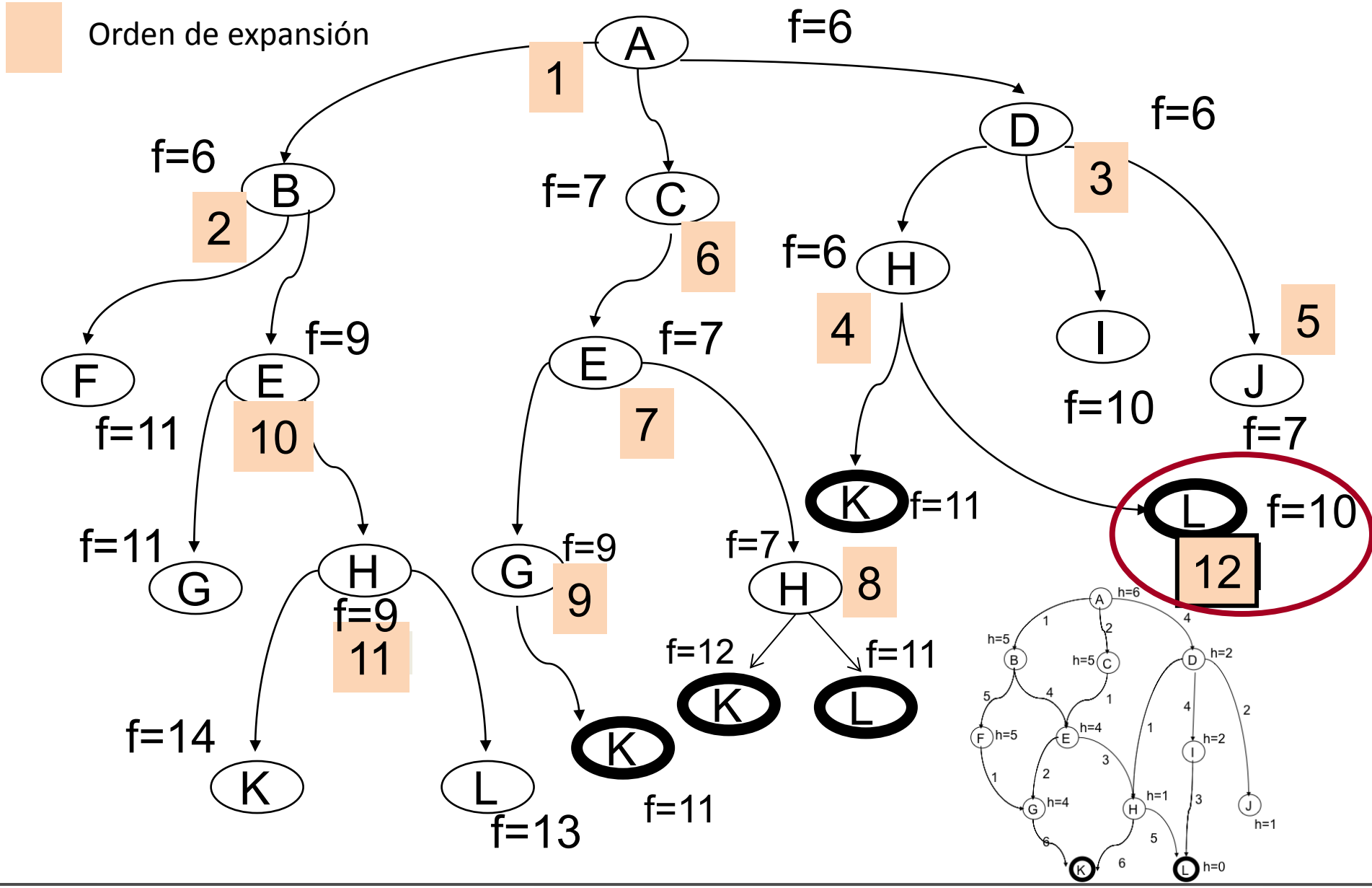
lista CLOSED = {Arad, Sibiu, Rimnicu Vilcea, Fagaras, Pitesti}

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.

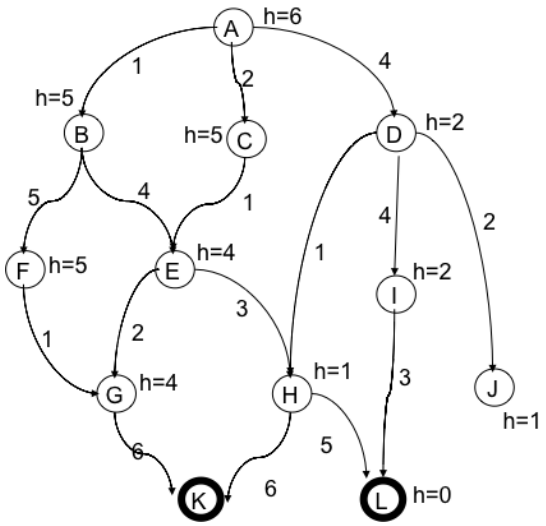
3. Búsqueda A*: otro ejemplo



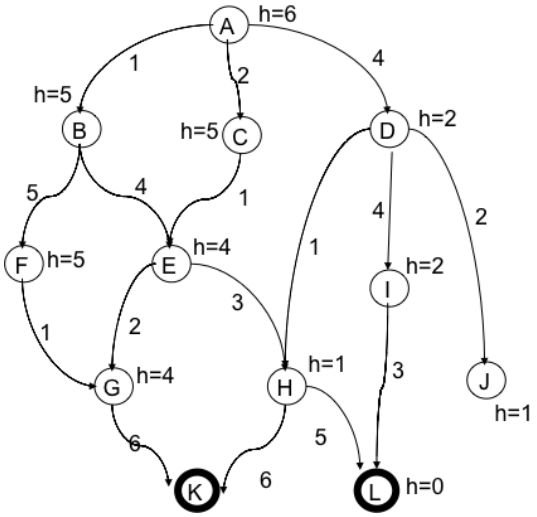
3. Búsqueda A*: versión TREE-SEARCH sin control de nodos repetidos



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



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.

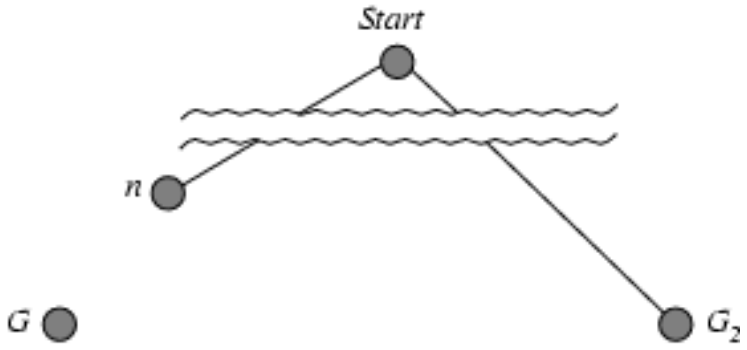
3. Búsqueda A*: optimalidad

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

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 . f(n) nunca sobreestima el coste del camino a través de n.
  - $f(n) = g(n)+h(n) \leq g(n)+h^*(n) = g(G) = f(G) \Rightarrow f(n) \leq g(G)$ (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 estado 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

3. Búsqueda A*: optimalidad

En un algoritmo A* GRAPH–SEARCH con re-expansión se garantiza que se encuentra la solución óptima si $h(n)$ es una heurística admisible.

En un algoritmo A* GRAPH–SEARCH sin re-expansión se 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 generado con una acción a se cumple $h(n) <= h(n')+c(n,a,n')$

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

Por tanto, el algoritmo GRAPH-SEARCH sin re-expansión también devuelve la solución óptima porque

- 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, se ha encontrado la solución óptima a dicho nodo, por lo que no es necesario volver a re-expandir n en caso de que sea un nodo repetido (el coste del nodo 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)
  - 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
  - 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

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

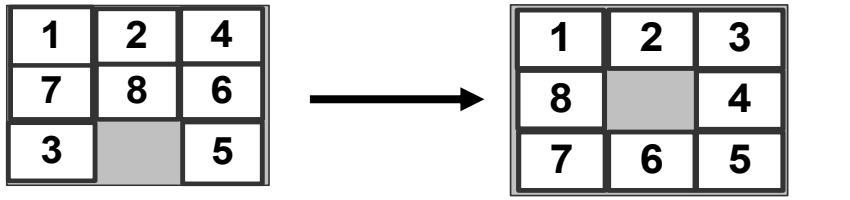
Considerar el problema con menos restricciones, obteniendo así otro problema que se resuelve con una complejidad menor que la del problema inicial.

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



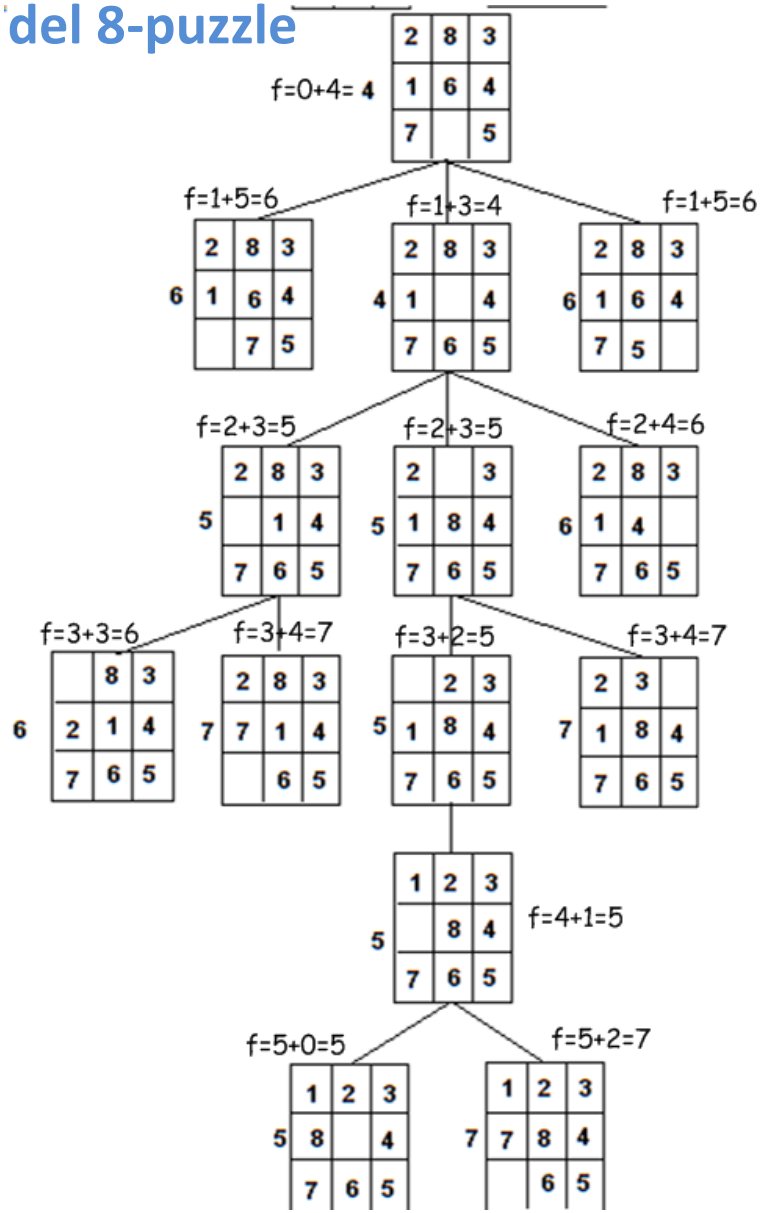
Estado inicial Estado objetivo

- 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): distancias de Manhattan (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), ∀n)

4.1. Heurísticas para el problema del 8-puzzle

Búsqueda A* con la heurística
fichas descolocadas

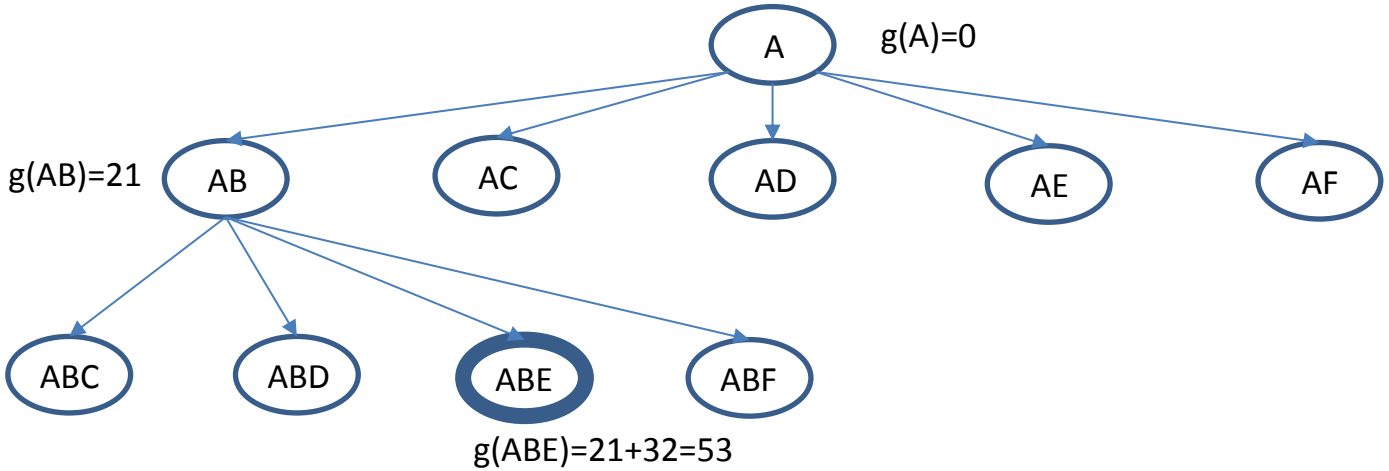


4.2. Heurísticas para el problema del viajante de comercio

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



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

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) →
C, D, F, volver a A

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) →
C, D, F, volver a A

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]

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

[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

5. Evaluación de funciones heurísticas

Dificultad de aplicar un análisis matemático

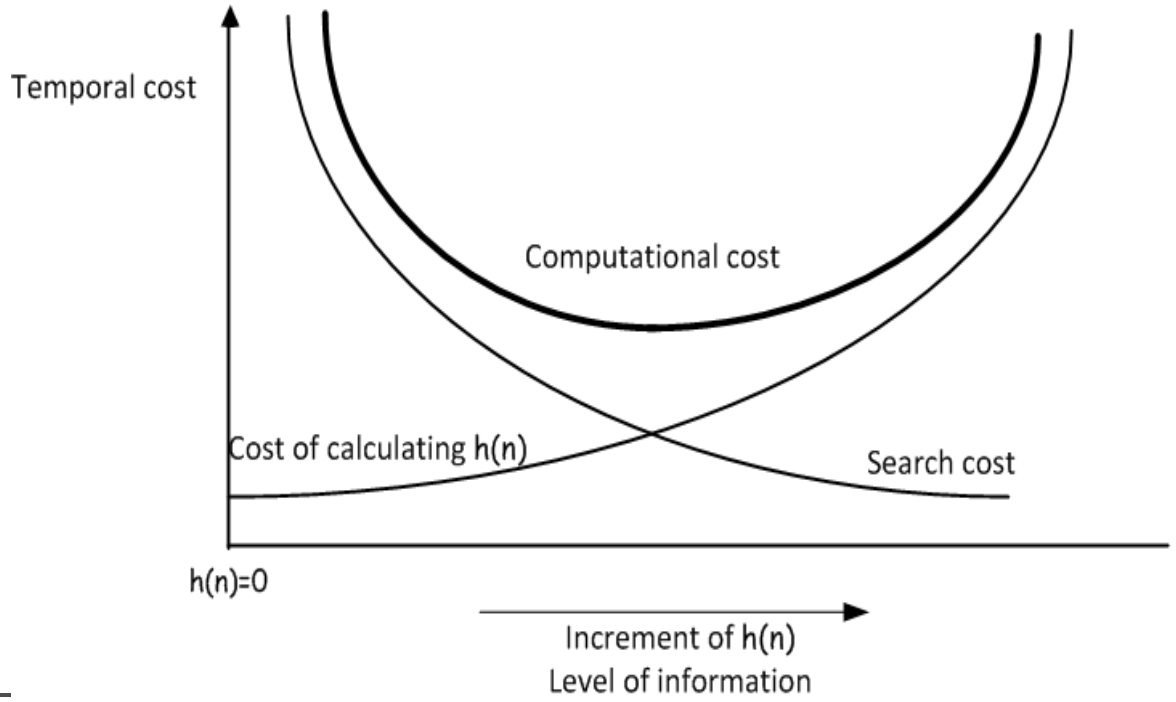
Se aplican métodos estadísticos/experimentales.

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

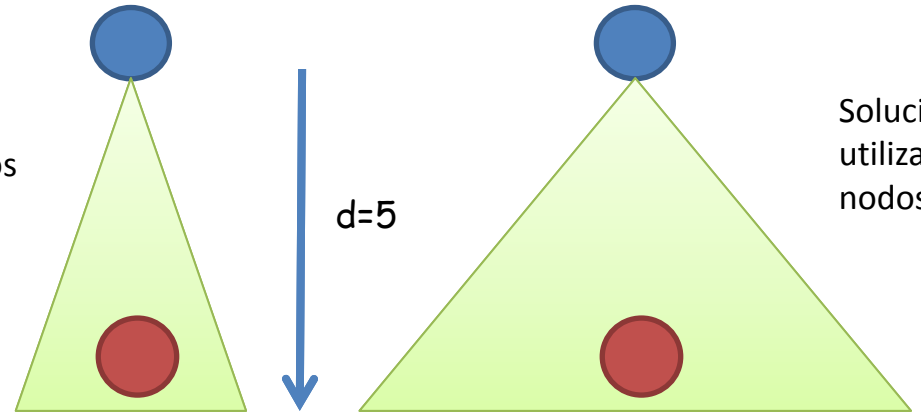
d = Profundidad de la solución

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.