Universidad Zaragoza
1542

Inteligencia Artificial (30223)

Lección 2. Búsqueda Informada

Curso 2022-2023

José Ángel Bañares
28/09/2022 Dpto. Informática e Ingeniería de Sistemas.

Universidad Zaragoza
1542

Índice

- Informada= Utiliza conocimiento problema
- Estrategias
  - Búsqueda primero el mejor y sus variantes (A*)
- Función heurística

Universidad Zaragoza
1542

Descripción informal de la búsqueda en grafo

function BÚSQUEDA-GRAFO(***problema***, ***estrategia***) **returns** **solución** o **fallo**
    Inicializa la ***frontera*** utilizando el estado inicial del problema
    Inicializa el conjunto de nodos **explorados** a vacío
    loop do
        if la ***frontera*** está vacía **then return *fallo***
        elige un ***nodo*** hoja de la frontera de acuerdo a una ***estrategia***
        if el ***nodo*** contiene un nodo objetivo **then return *solución***
        añade el ***nodo*** al conjunto de nodos ***explorados***
        expande el nodo elegido, añadiendo los nodos resultantes a la frontera
            sólo si no está en la ***frontera*** o en el conjunto ***explorados***

Una estrategia se define por el orden de expansión de nodos

Búsqueda primero el mejor

Idea: Utilizar una **función de evaluación** de **cada nodo** f(n) que **estima** como de "**prometedor**" es el nodo.

Aproximación general de la búsqueda informada

- **f(n)** es una estimación del coste. Elegiremos el nodo con menor coste estimado basado en la función **f(n)**
- La elección de **f(n)** determina la **estrategia**
- La mayoría de los algoritmos primero el mejor incluyen una función heurística, que denominaremos **h(n).**

h(n) = *estimación del coste del camino menos costoso desde el estado en el nodo n al objetivo. Depende sólo del* ***estado*** *en n.*

Universidad Zaragoza

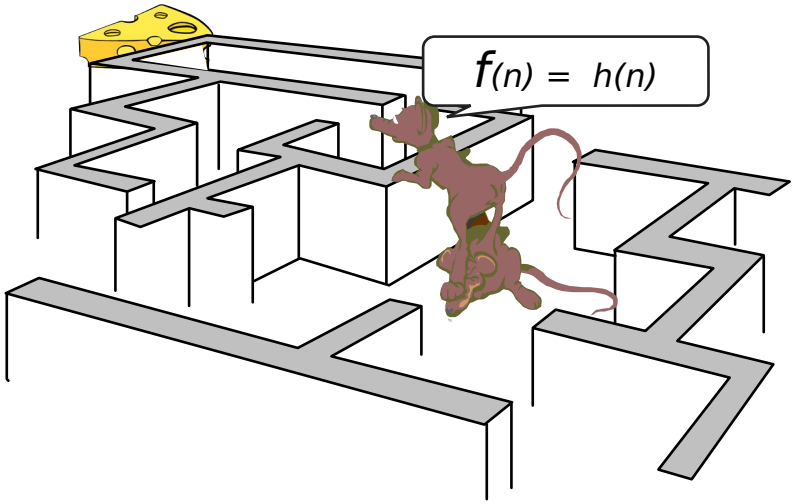
Búsqueda primero el mejor

La carga de **conocimiento** del problema viene dada por **h(n)**

Algoritmo idéntico a **coste uniforme**, salvo estrategia que **utiliza f(n) = h(n)** en lugar de g(n) donde g(n) es el coste para llegar al nodo nodo n.

h(n) es sólo función del estado.

- **h(n)** estima cuanto cuesta llegar al objetivo desde el nodo n al objetivo por el camino de menor coste.
- **h(n)** sólo estima lo prometedor del nodo en función del estado n.
- **h(n)** es una función **no negativa**, específica del problema, y con **h(n)=0** si el estado del nodo es un objetivo.

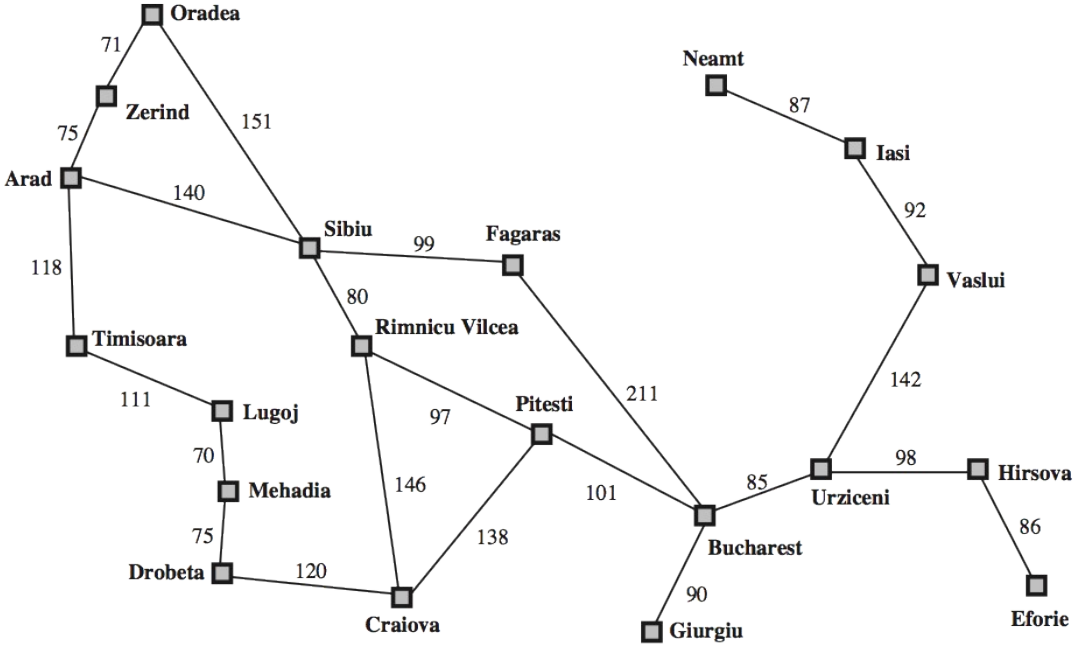


- Best-first search
- Greedy best-search

Universidad Zaragoza

Ejemplo heurística: Distancia en línea recta al objetivo

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



- h_{SLD}= heurística de la línea recta al objetivo / **straight-line distance heuristic.**

Universidad Zaragoza
1542

Algoritmo voraz /Greedy algorithm

- Evaluación de función $f(n) = h(n)$ (heurística)

  = estimación del coste de n al objetivo más próximo

- El algoritmo voraz expande el nodo que parece estar más próximo al objetivo.

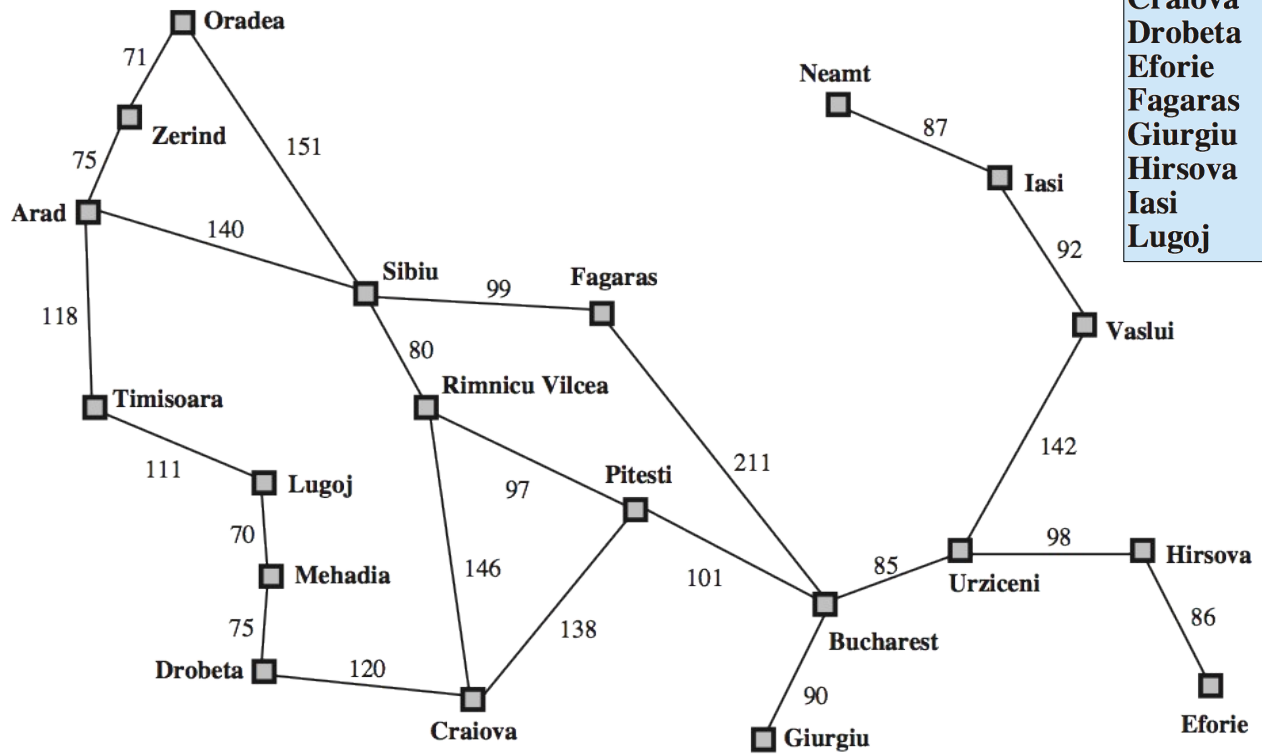
  P. e. $h_{SLD}(n)$ = distancia en línea recta de n a Bucharest

Universidad Zaragoza

Ejemplo Algoritmo voraz

Queremos viajar de Arad a Bucharest.

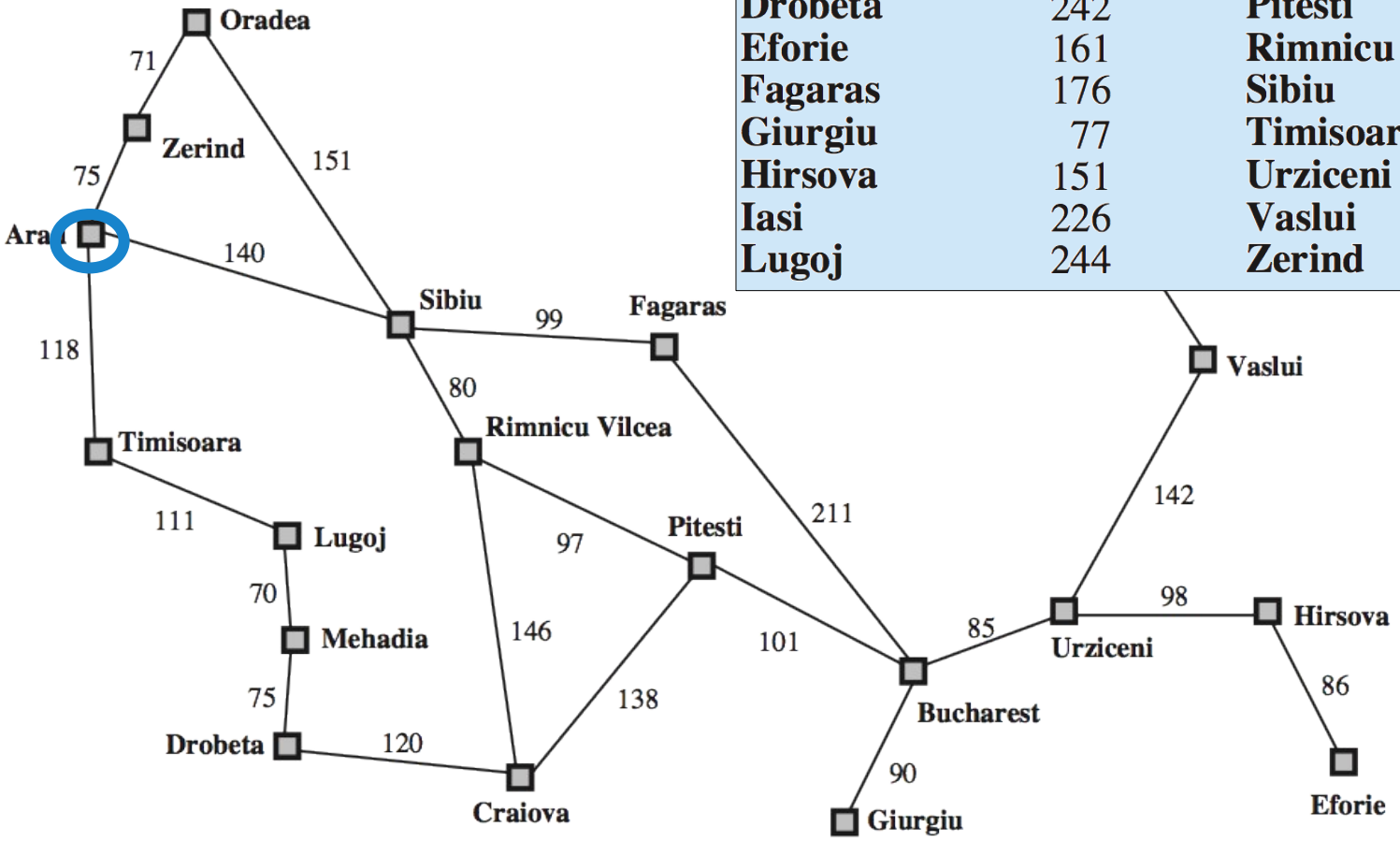
- Estado Inicial = Arad



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

Ejemplo Algoritmo voraz (en árbol)

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

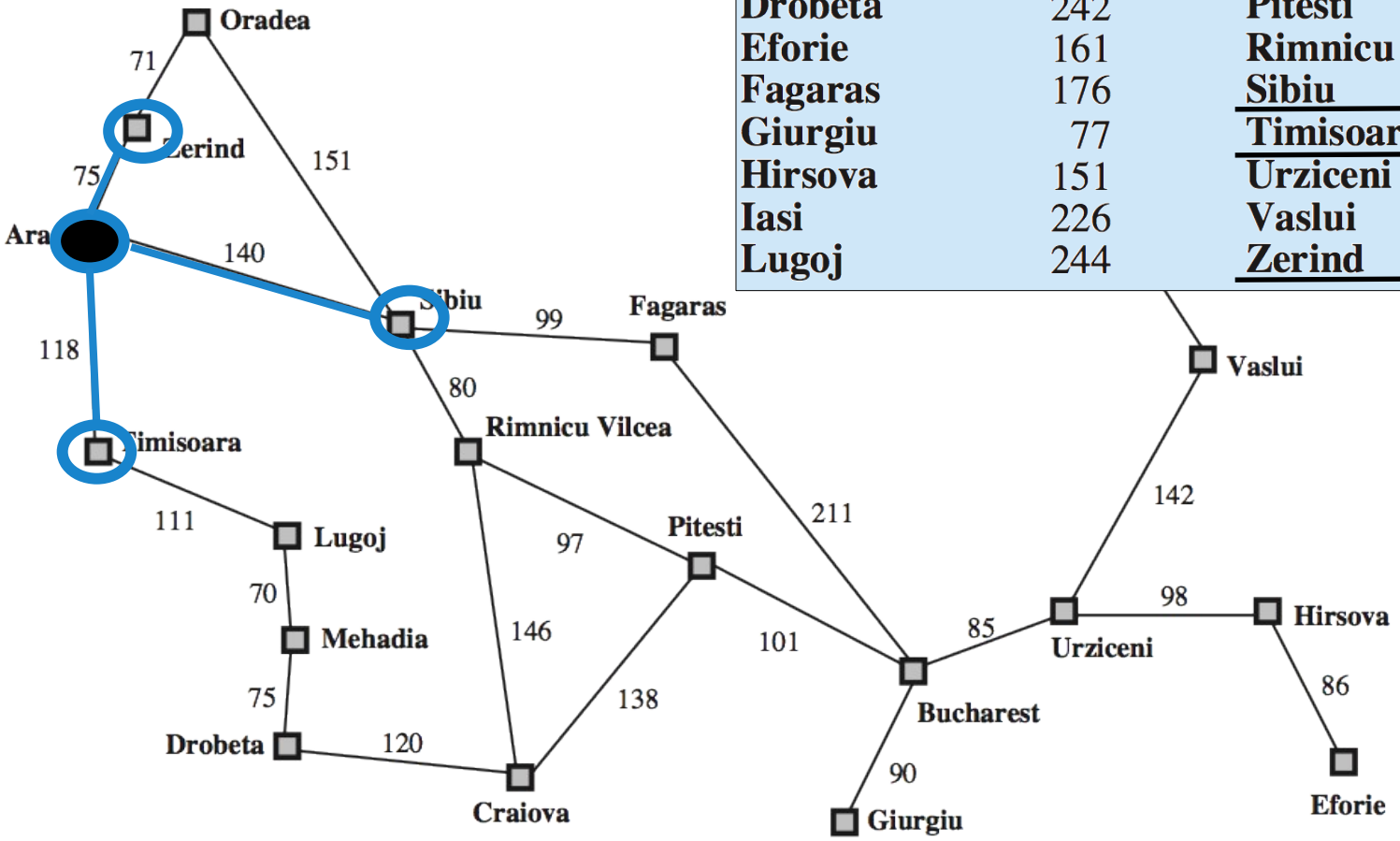


Ejemplo Algoritmo voraz

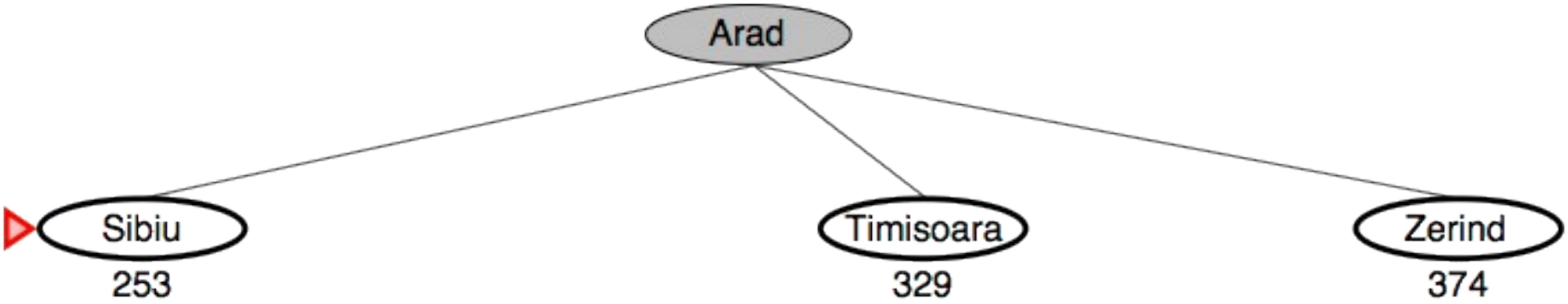
Universidad Zaragoza

Ejemplo Algoritmo voraz

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



Ejemplo Algoritmo voraz

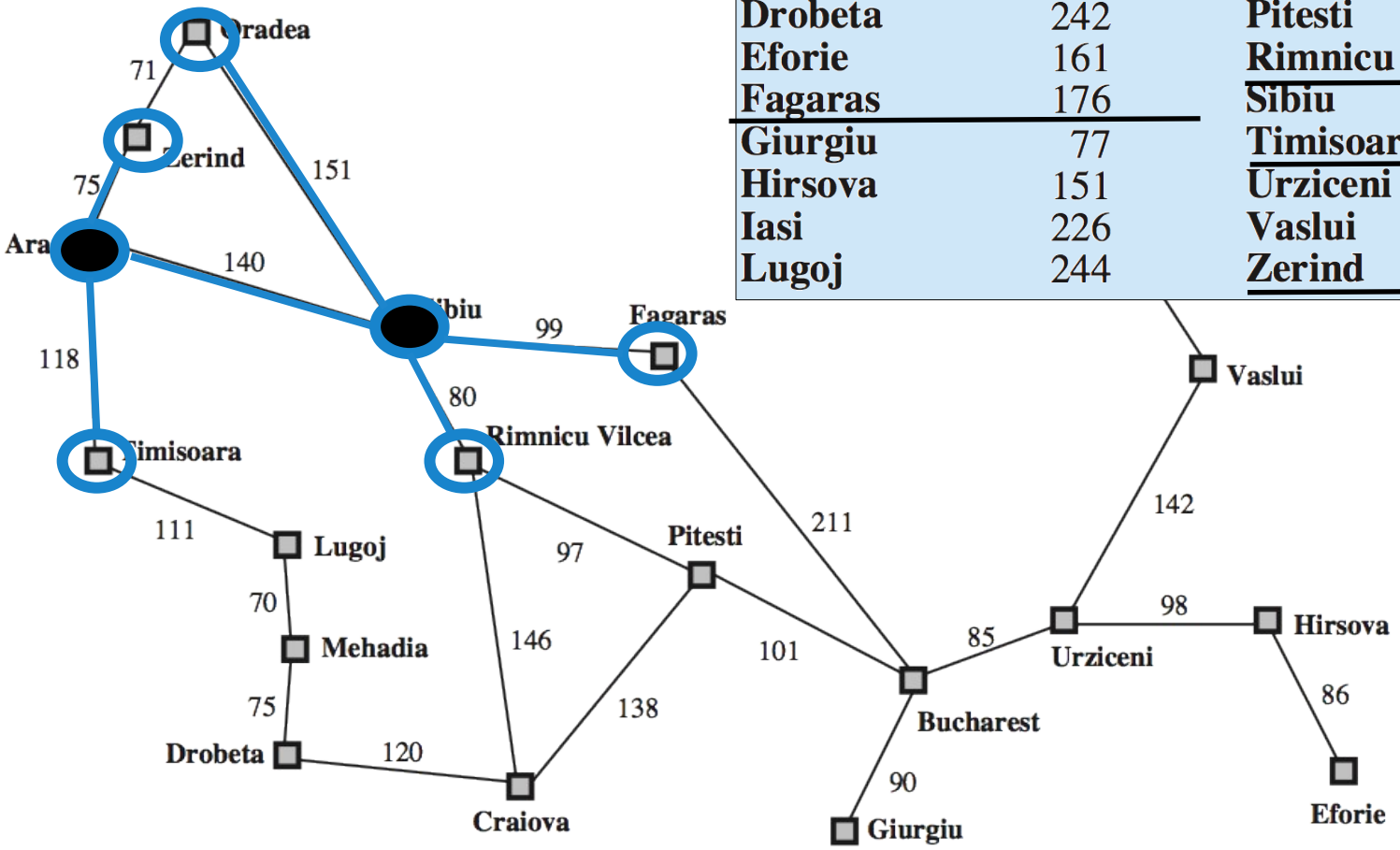


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

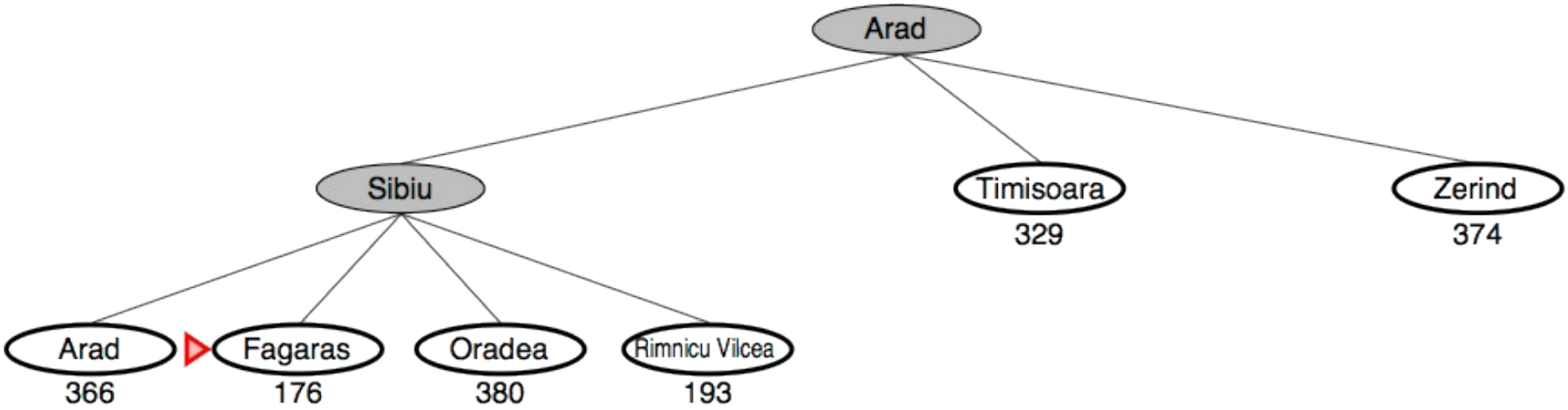
Universidad Zaragoza

Ejemplo Algoritmo voraz (en árbol)

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



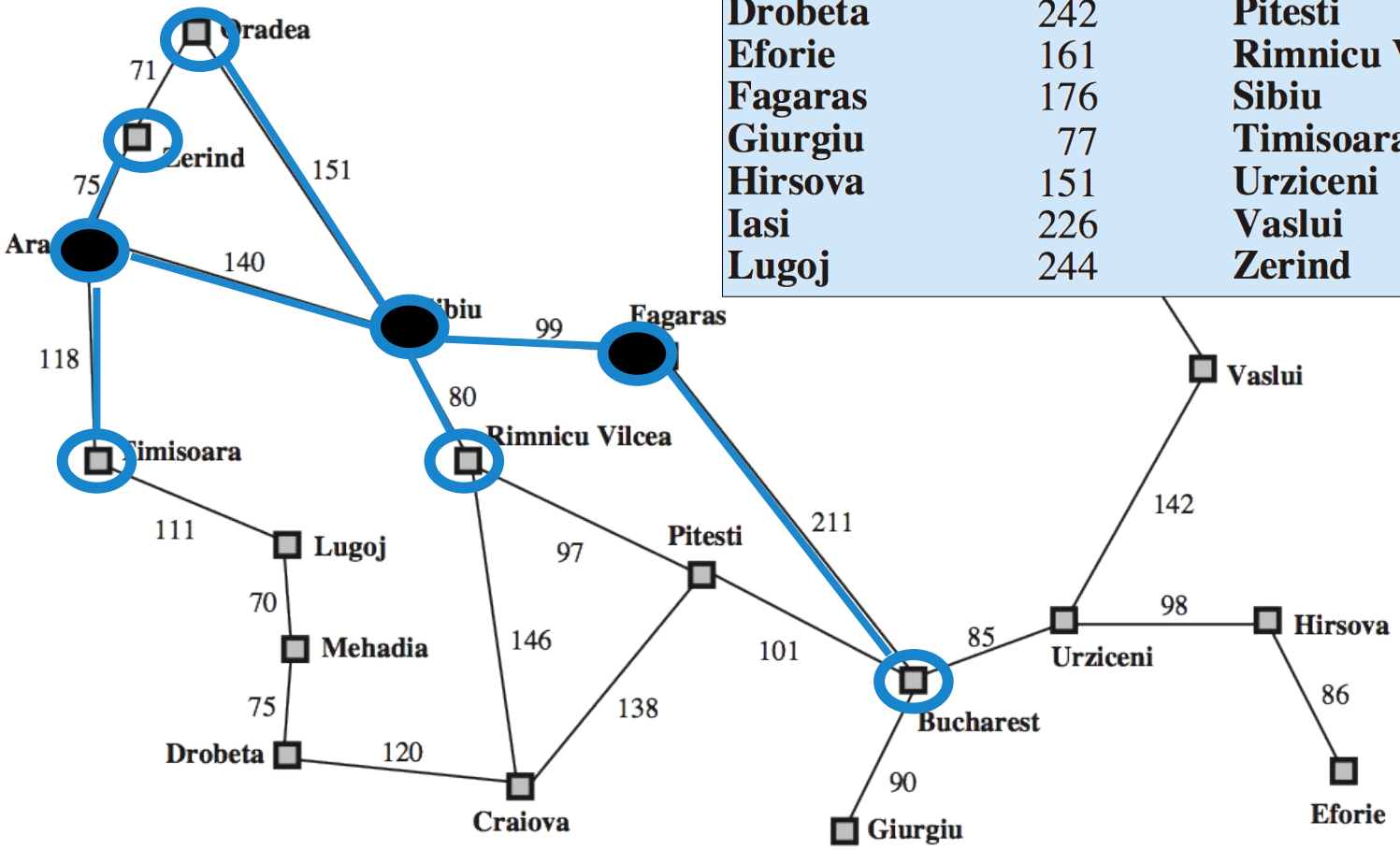
Ejemplo Algoritmo voraz (en árbol)



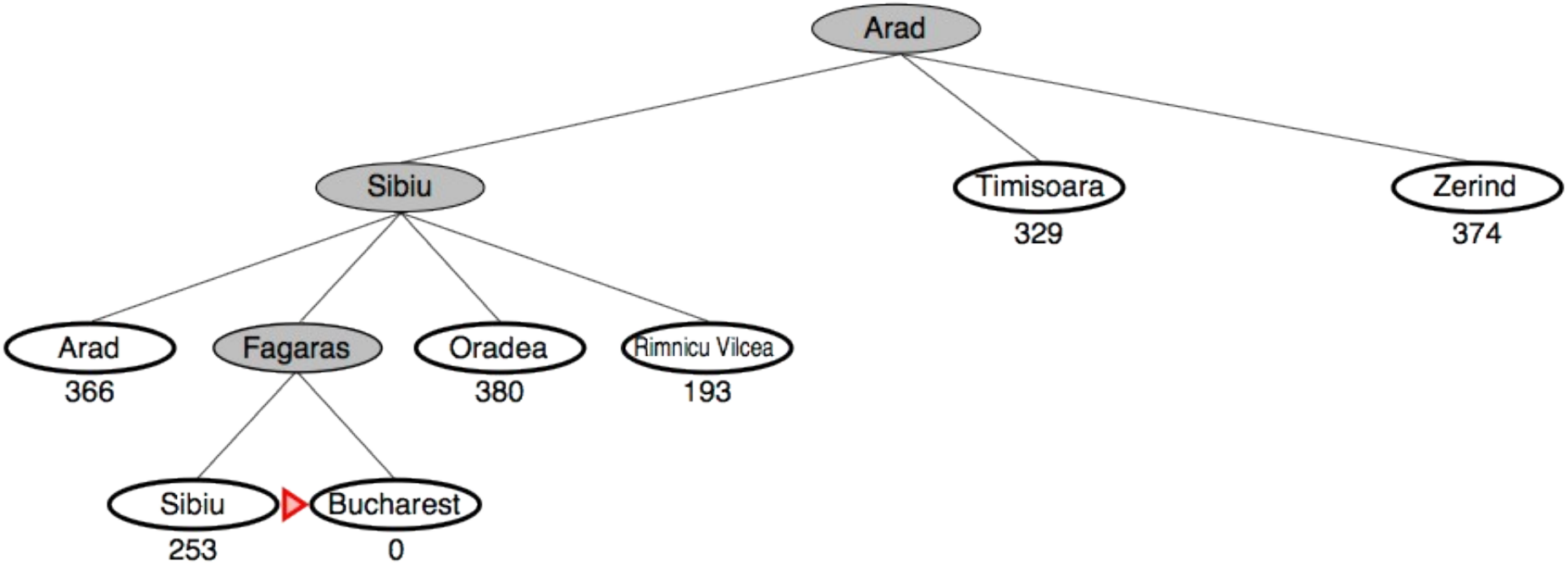
Universidad Zaragoza

Ejemplo Algoritmo voraz (en árbol)

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



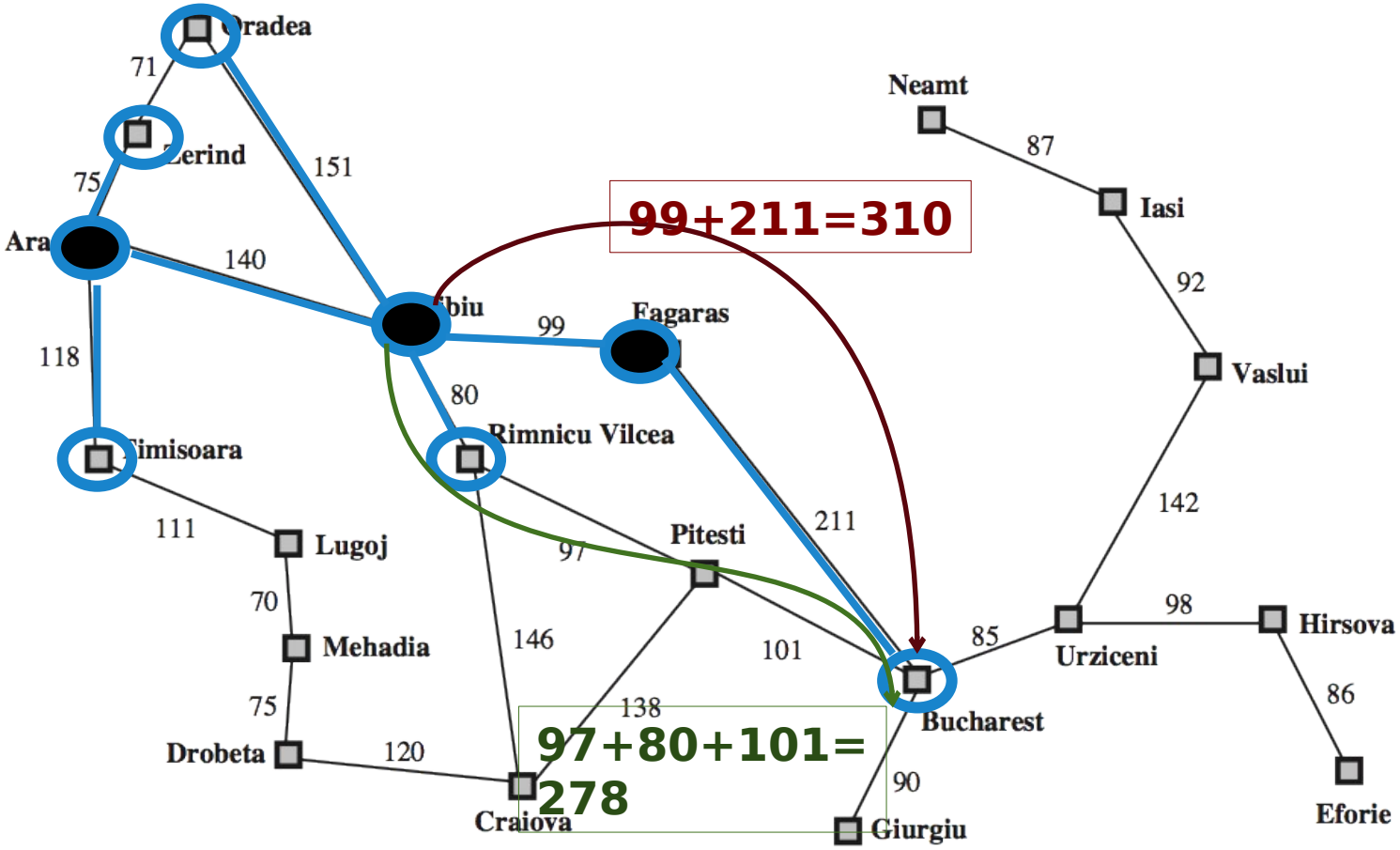
Ejemplo Algoritmo voraz (en árbol)



Universidad Zaragoza

Evaluación: ¿Óptimo?

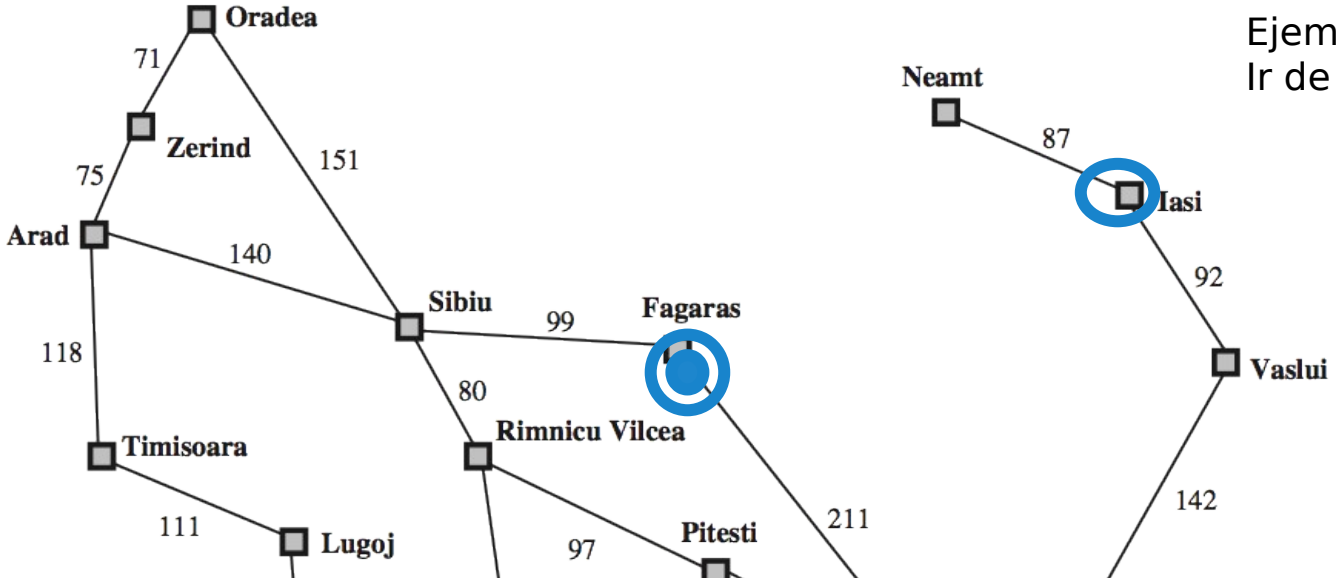
- Óptimo: NO.



Universidad Zaragoza

Evaluación algoritmo voraz

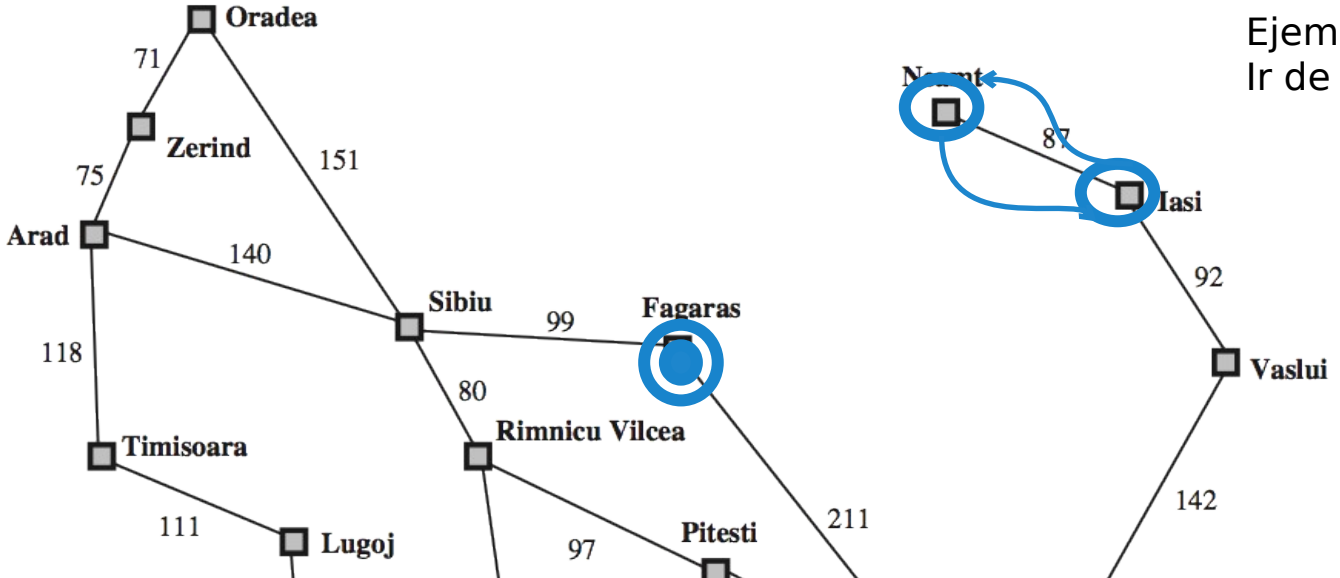
- Óptimo: NO
- Completo:
  - No en la búsqueda en árbol. Incluso con un espacio de estados finito, puede entrar en un bucle infinito.



Ejemplo Problema:
Ir de isasi a Fagaras

Evaluación algoritmo voraz

- Óptimo: NO
- Completo:
  - No en la búsqueda en árbol. Incluso con un espacio de estados finito, puede entar en un bucle infinito.

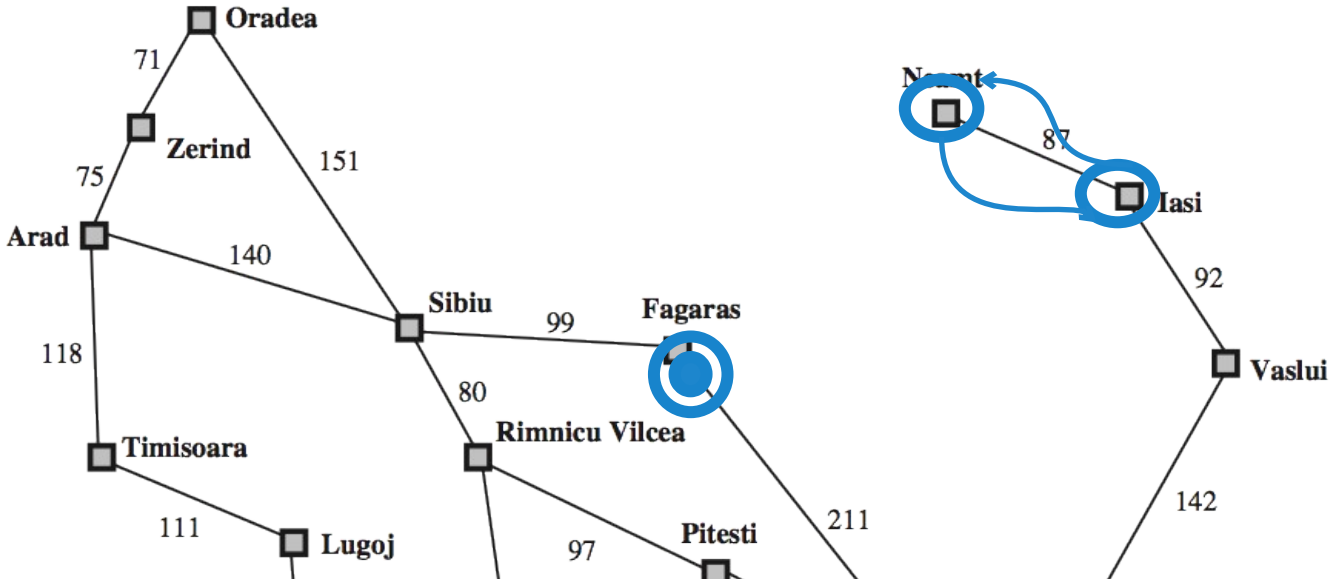


Ejemplo Problema:
Ir de isasi a Fagaras

Universidad Zaragoza

Evaluación algoritmo voraz

- Óptimo: NO
- Completo:
  - No en la búsqueda en árbol.
  - Si en búsqueda en grafo si el espacio es finito.



Evaluación algoritmo voraz

- Óptimo: NO. Como la búsqueda en profundidad.
- Completo:
  - No en la búsqueda en árbol.
  - Si en búsqueda en grafo si el espacio es finito.
- ¿Complejidad temporal? $O(b^m)$
  - El peor caso como la búsqueda en profundidad (con m la profundidad máxima del espacio de estados)

Evaluación algoritmo voraz

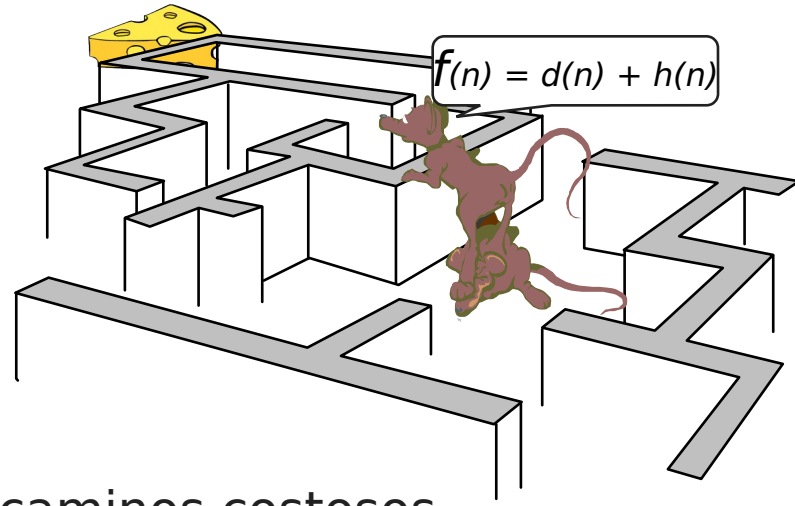
- Óptimo: NO. Como la búsqueda en profundidad.
- Completo:
  - No en la búsqueda en árbol.
  - Si en búsqueda en grafo si el espacio es finito.
- Complejidad temporal: $O(b^m)$
  - El peor caso como la búsqueda en profundidad (con m la profundidad máxima del espacio de estados)
- Complejidad espacial: $O(b^m)$
  - Guarda todo los nodos

Evaluación algoritmo voraz

- Óptimo: NO.
- Completo:
  - No en la búsqueda en árbol.
  - Si en búsqueda en grafo si el espacio es finito.
- Complejidad temporal: $O(b^m)$

  (con m la profundidad máxima del espacio de estados)
- Complejidad espacial: $O(b^m)$
- Una **buena heurística puede reducir la complejidad**
  - La reducción dependerá del problema y de la calidad de la heurística.

Universidad Zaragoza



Búsqueda A*

Idea: evitar expandir nodos que tienen caminos costosos.

- Función de evaluación *f(n)=g(n) + h(n)*

  g(n)=coste para alcanzar el nodo *n*

  (coste real desde estado inicial al nodo n)

  h(n)=estimación del coste para alcanzar el objetivo desde el nodo *n*.

  (estimación del coste desde nodo n al objetivo)

  f(n)= estimación del coste total del camino desde *n* al objetivo.

  (estimación del camino, incluyendo lo que me ha costado llegar a n desde el estado inicial, y una estimación de lo que me costará llegar de n al objetivo)

Algoritmo idéntico a la búsqueda de coste uniforme, utilizando ***f*** en lugar de ***g***.

Búsqueda A*

La búsqueda A* es óptima y completa si la función heurística cumple ciertas propiedades (**admisibilidad en árbol** y **consistencia en grafo**)

- Una heurística es **admisible** si nunca sobreestima el coste real de alcanzar el objetivo.
- Una heurística admisible es **optimista**

Formalmente:

1. ***h(n) <= h*(n)*** donde ***h*(n)*** es el coste **real** para llegar al objetivo desde n
2. ***h(n) >= 0*** , de forma que ***h(G)=0*** para cualquier objetivo

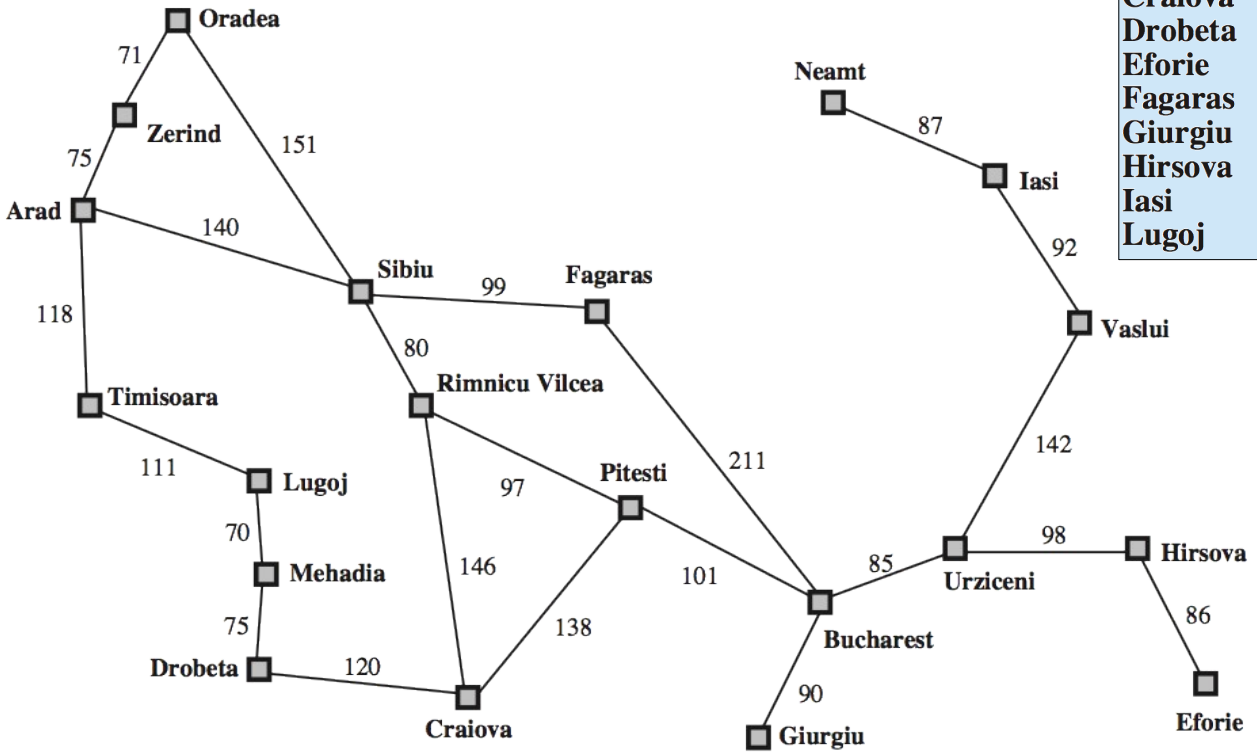
p.e. $h_{SLD}(n)$ nunca sobreestima la distancia real a recorrer.

Universidad Zaragoza

Ejemplo Rumanía: A*(árbol)y h_{SLD}

Queremos viajar de Arad a Bucharest.

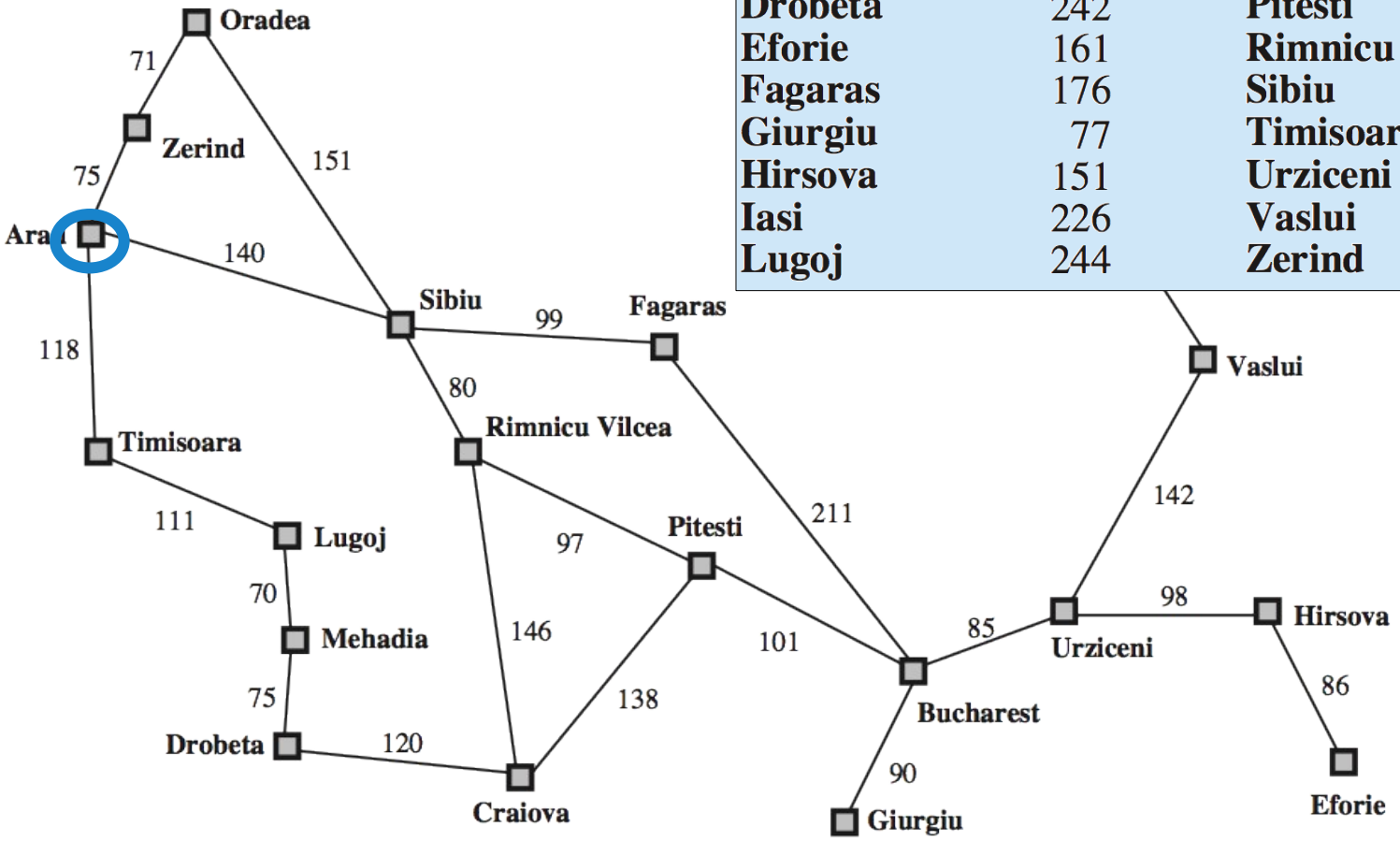
- Estado Inicial = Arad



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

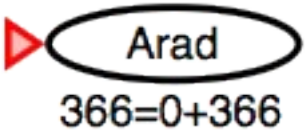
Ejemplo Rumanía: A* (árbol) y h_{SLD}

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



Universidad Zaragoza
1542

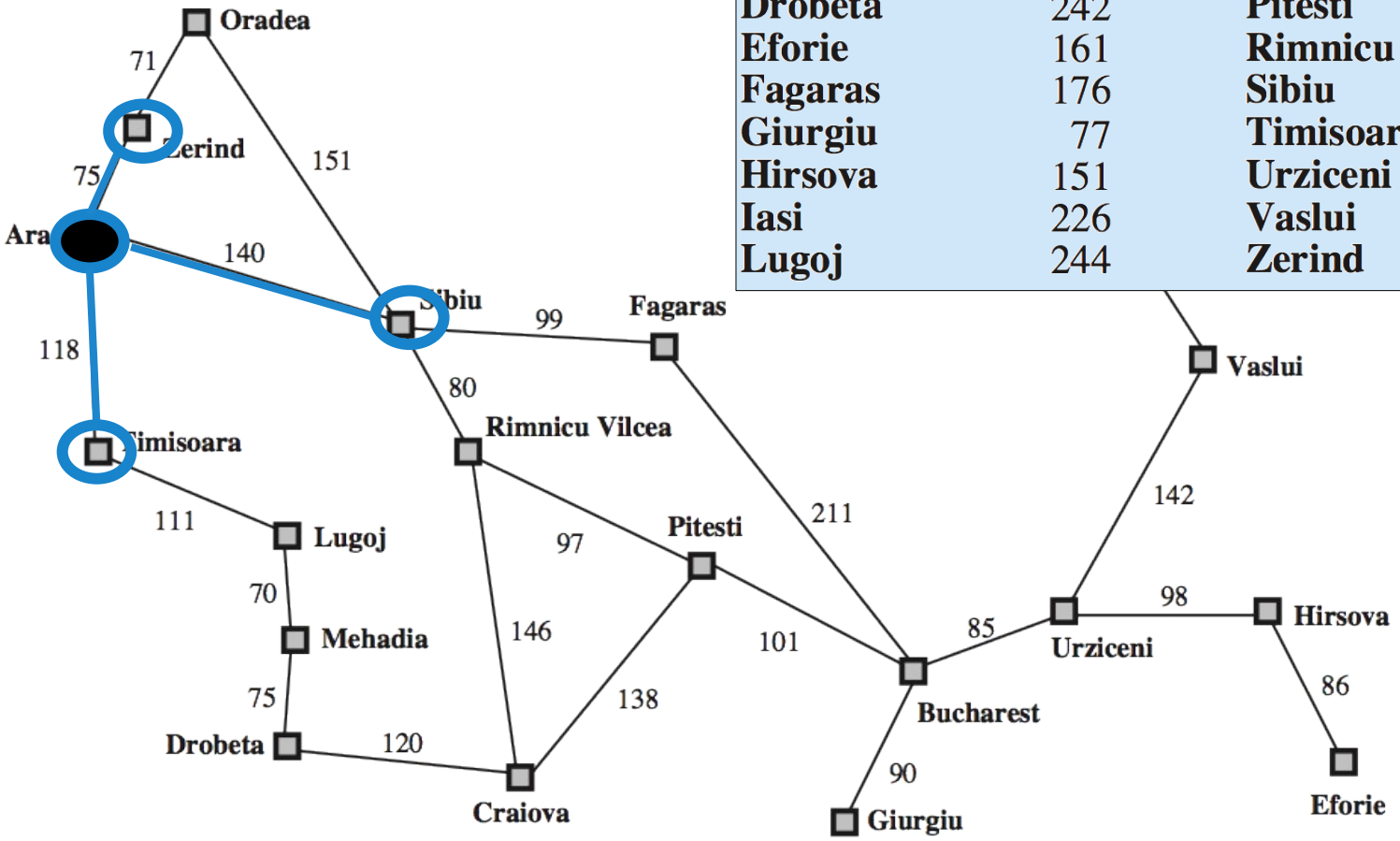
Ejemplo Rumanía: A* y h_{SLD}



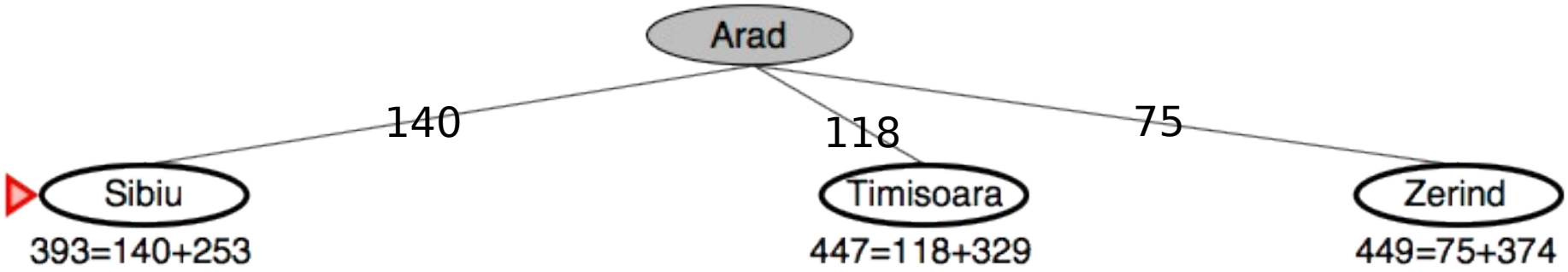
Universidad Zaragoza

Ejemplo Rumanía: A* y h_{SLD}

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



Ejemplo Rumanía: A* y h_{SLD}

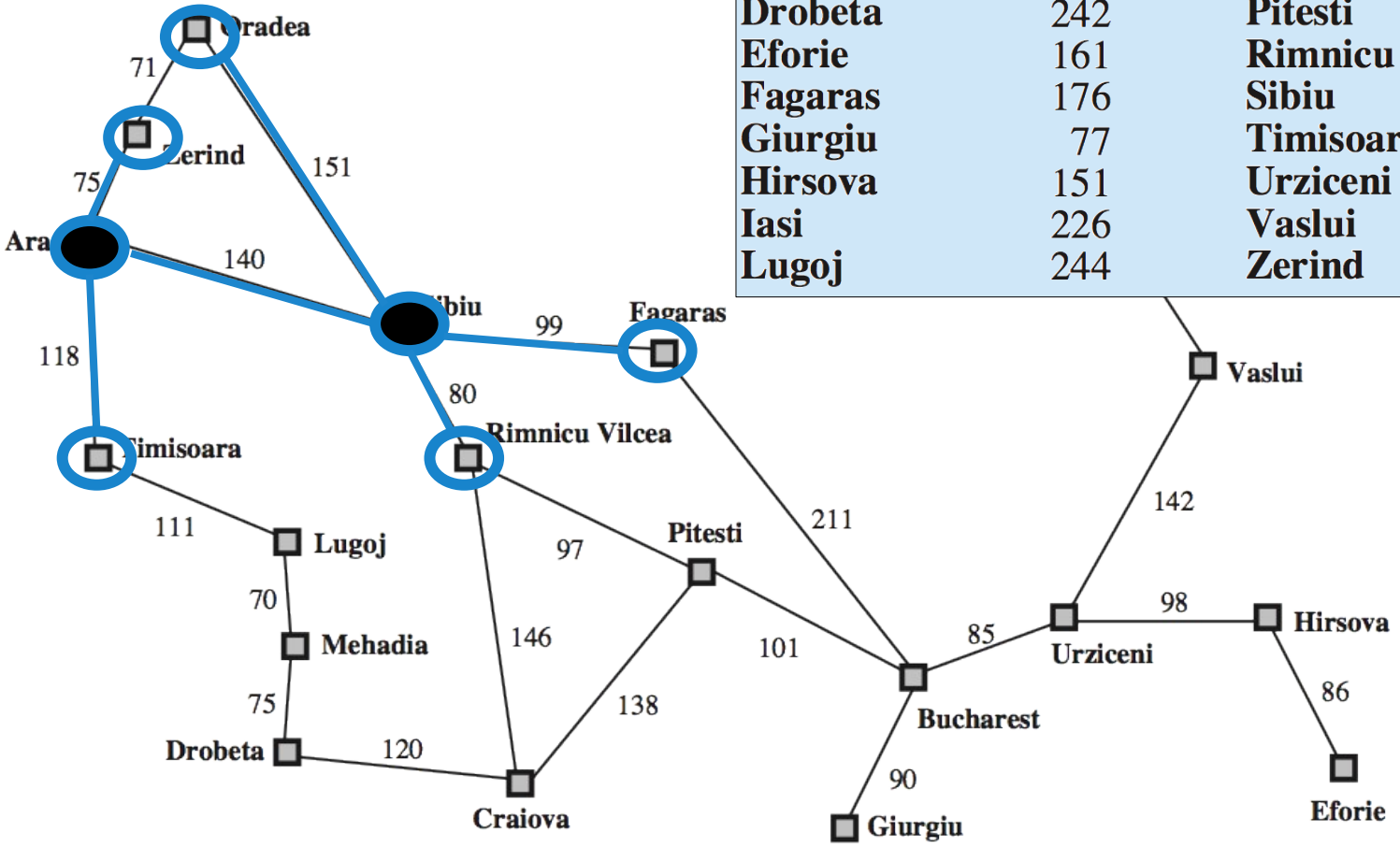


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

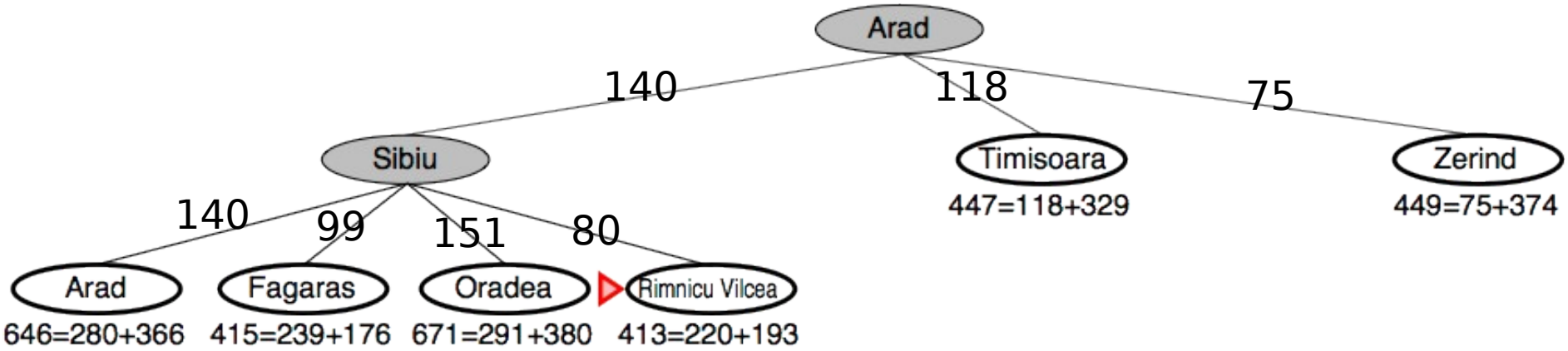
Universidad Zaragoza

Ejemplo Rumanía: A* y h_{SLD}

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



Ejemplo Rumanía: A* y h_{SLD}

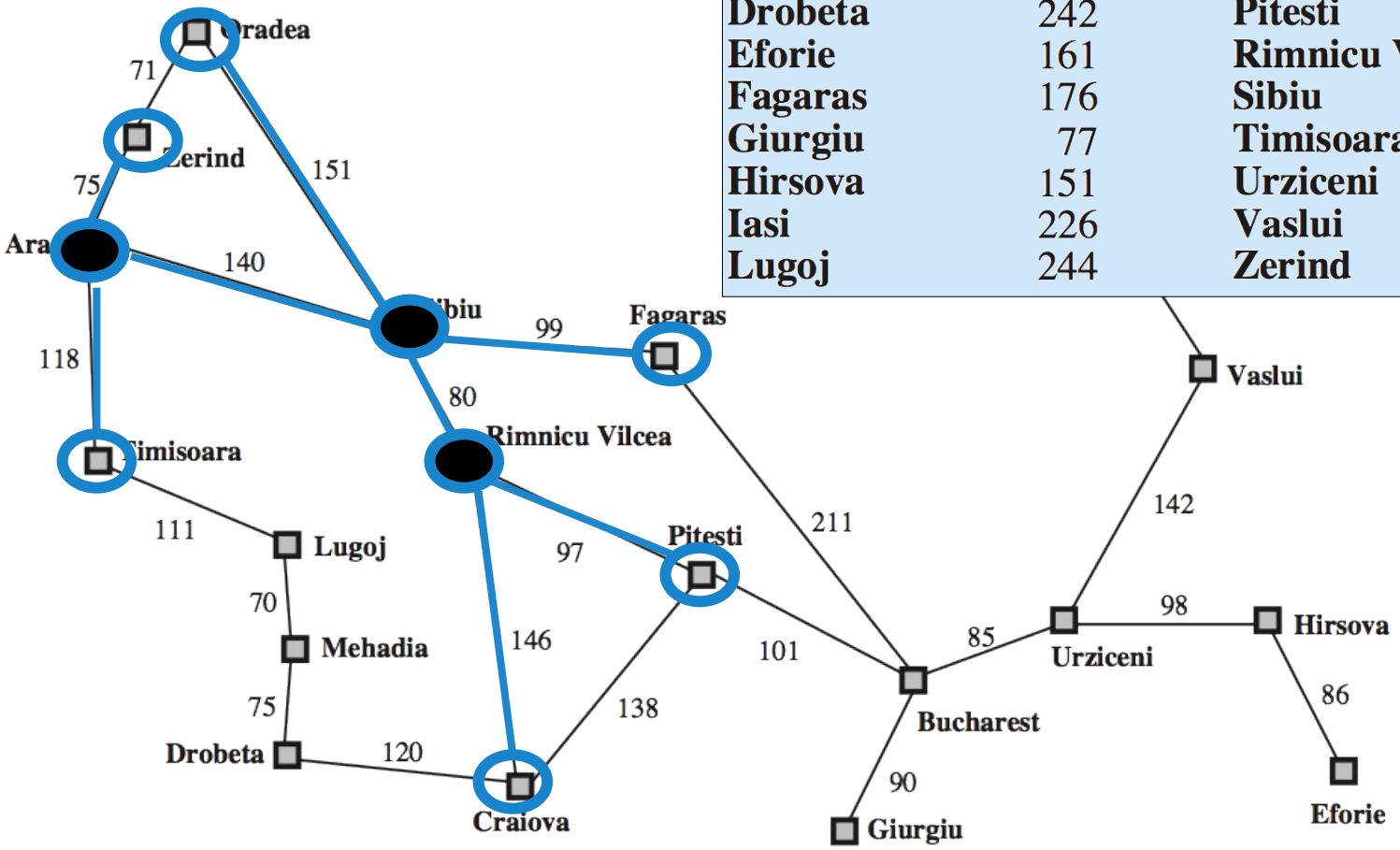


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Universidad Zaragoza

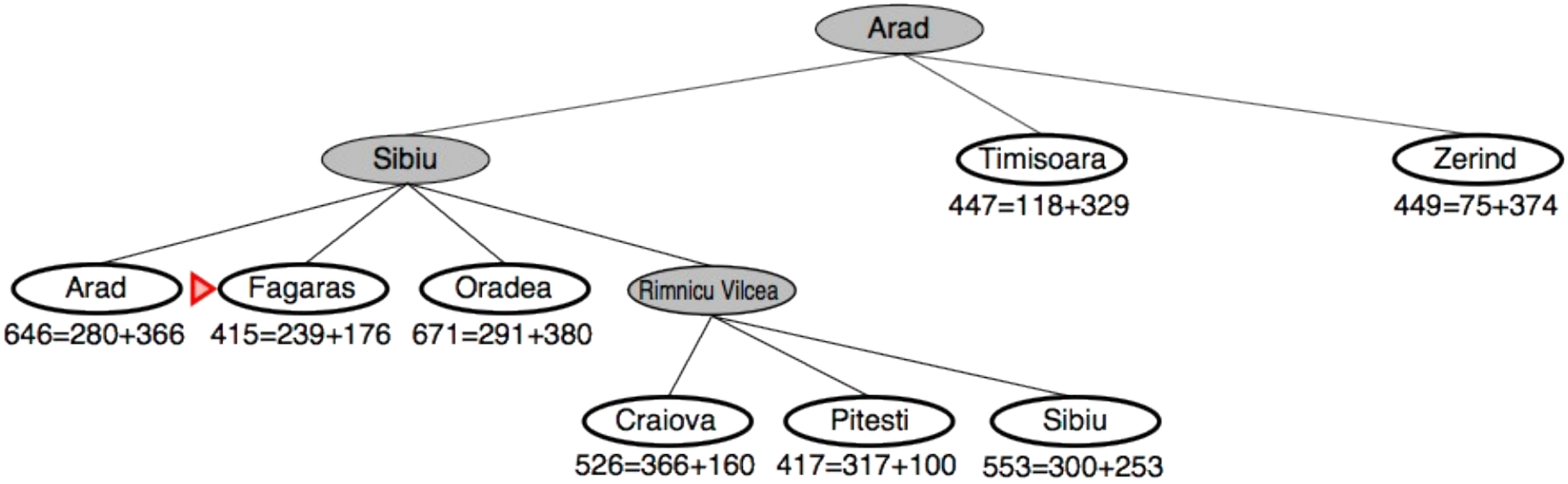
Ejemplo Rumanía: A* y h_{SLD}

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



Universidad Zaragoza

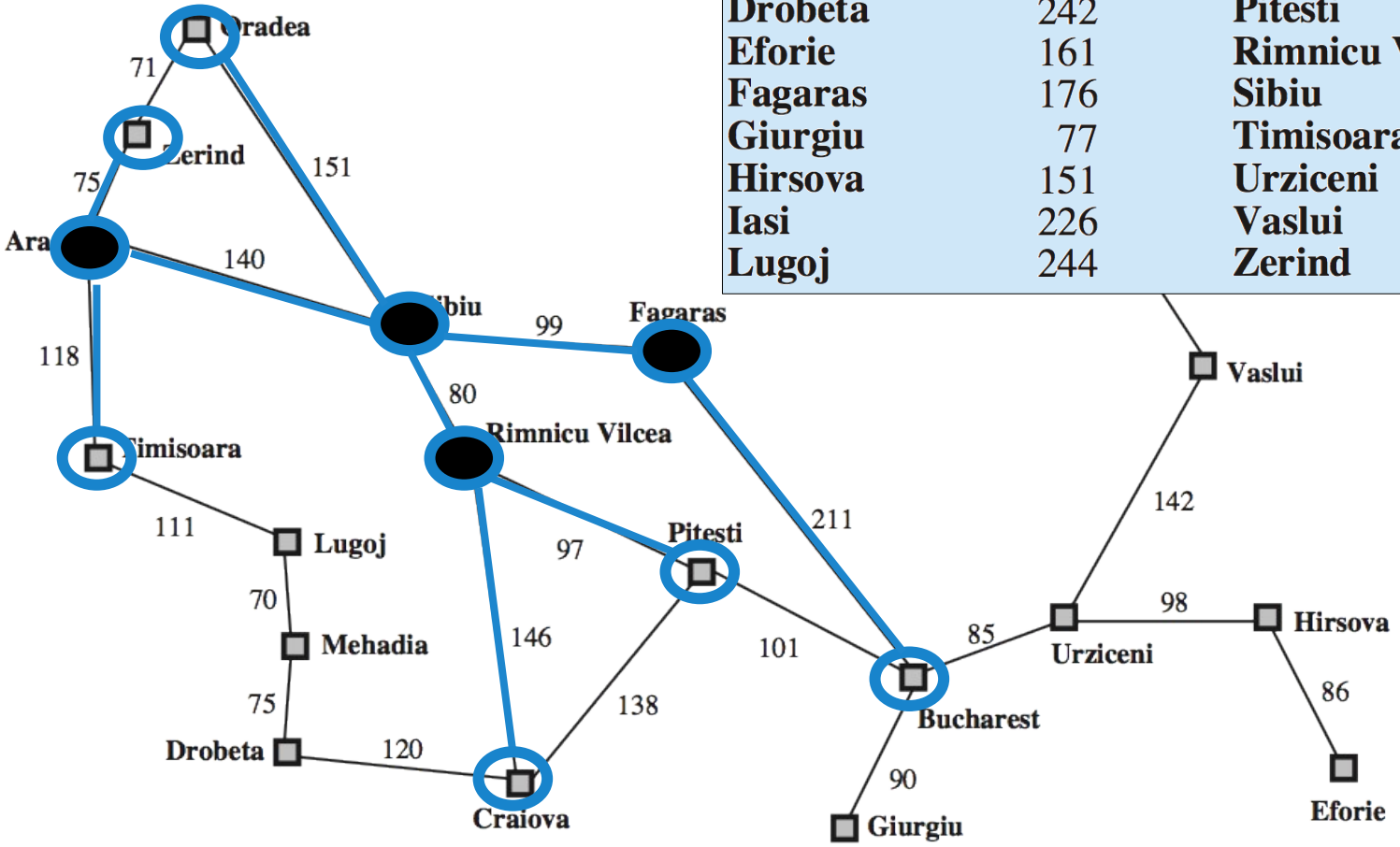
Ejemplo Rumanía: A* y h_{SLD}



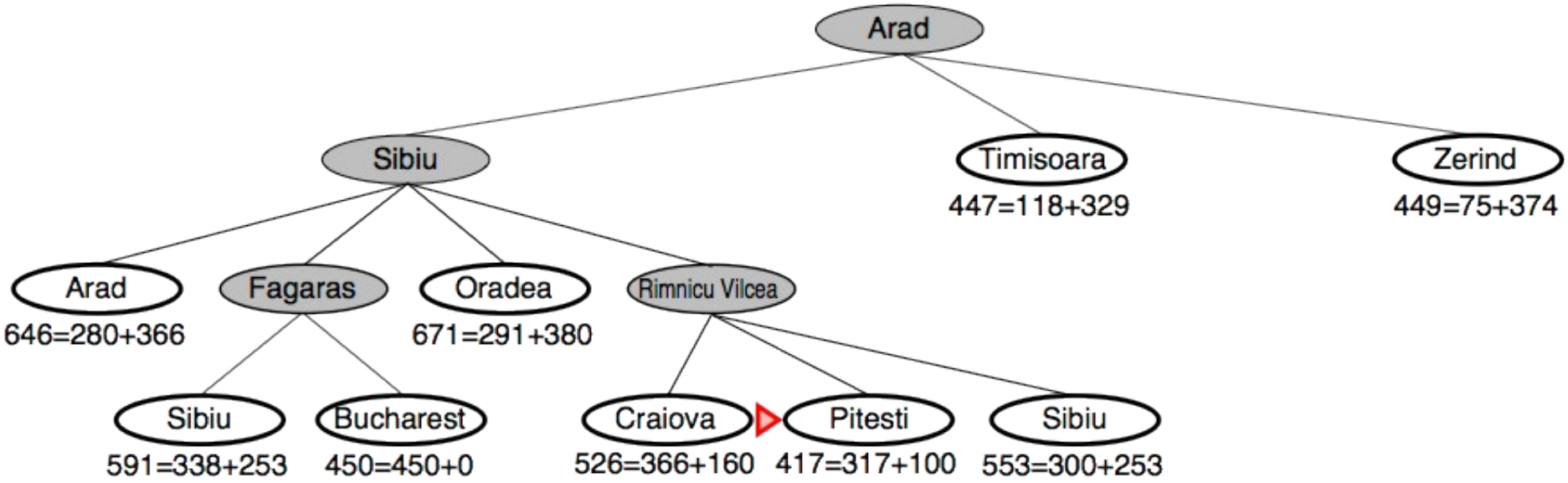
Universidad Zaragoza

Ejemplo Rumanía: A* y h_{SLD}

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

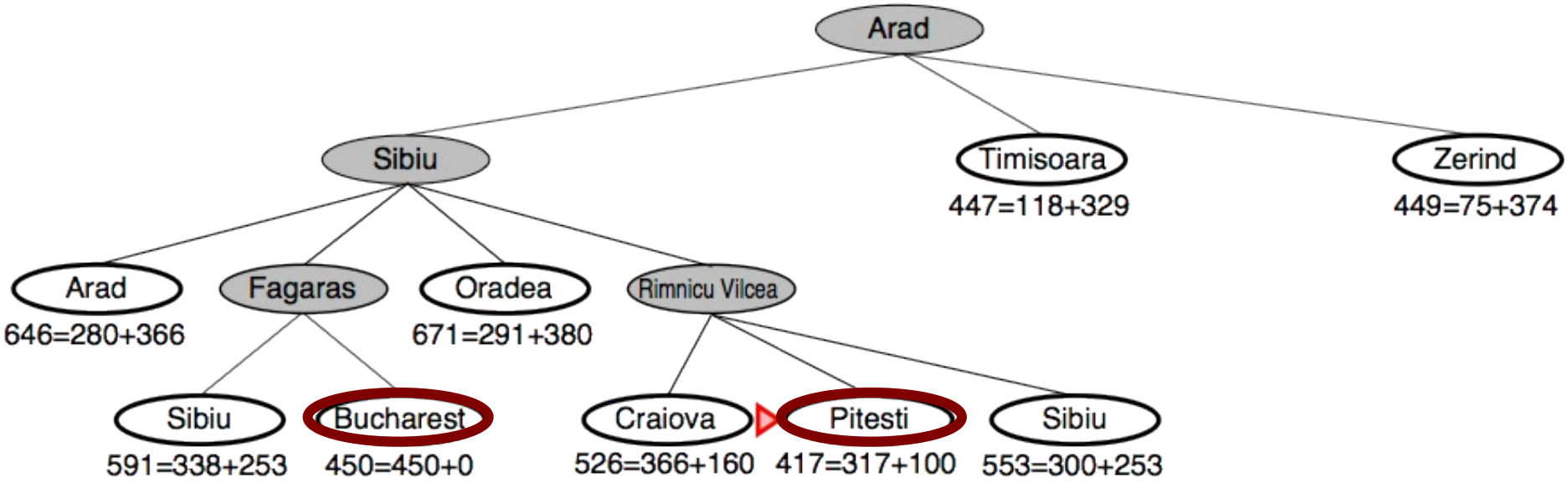


Ejemplo Rumanía: A* y h_{SLD}



Universidad Zaragoza

Ejemplo Rumanía: A* y h_{SLD}

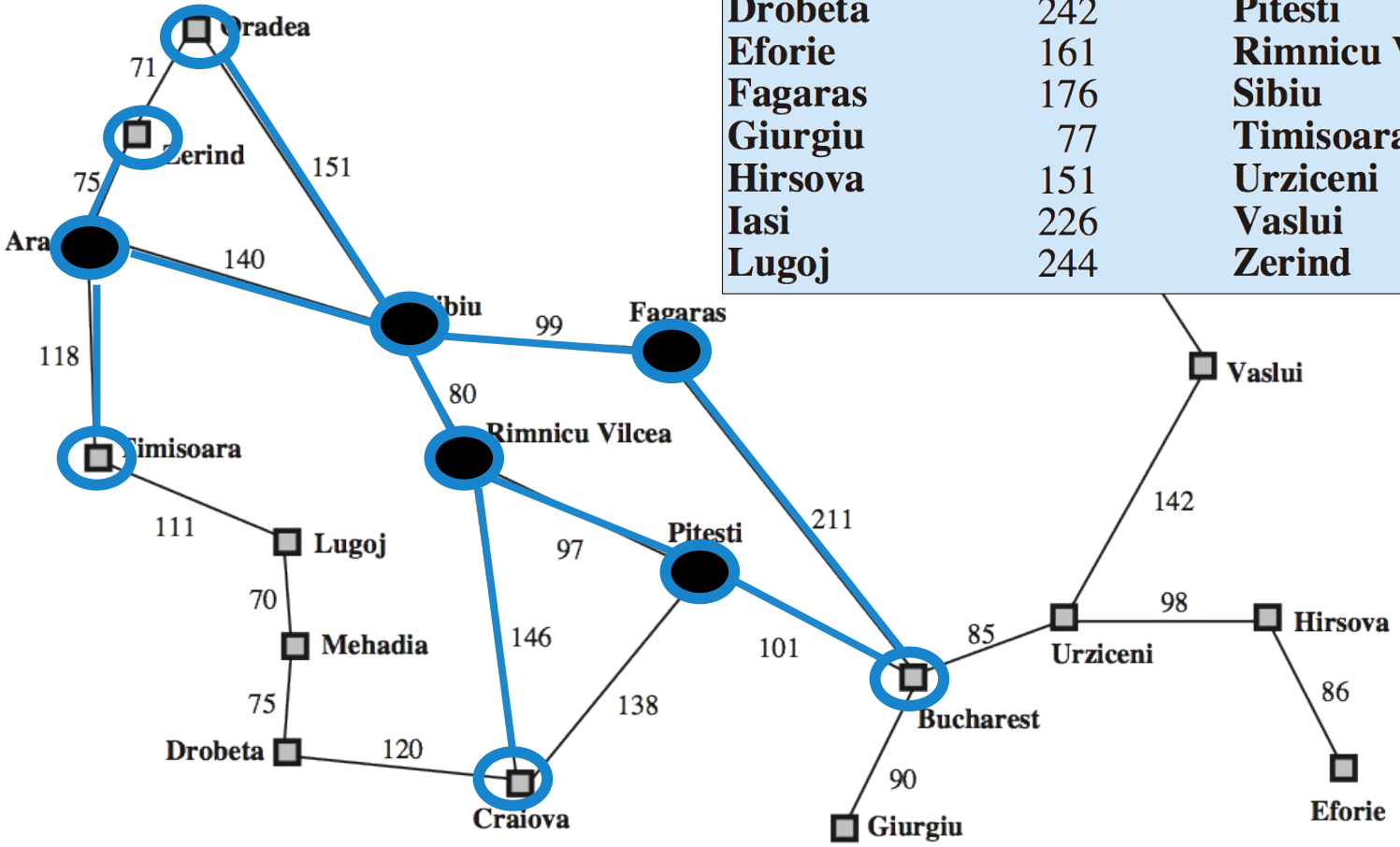


Aparece Bucharest en la frontera, pero con f= 450.
Pitesi tiene el menor valor de f 417,
podríamos encontrar una solución de menor coste a través de Pitesi

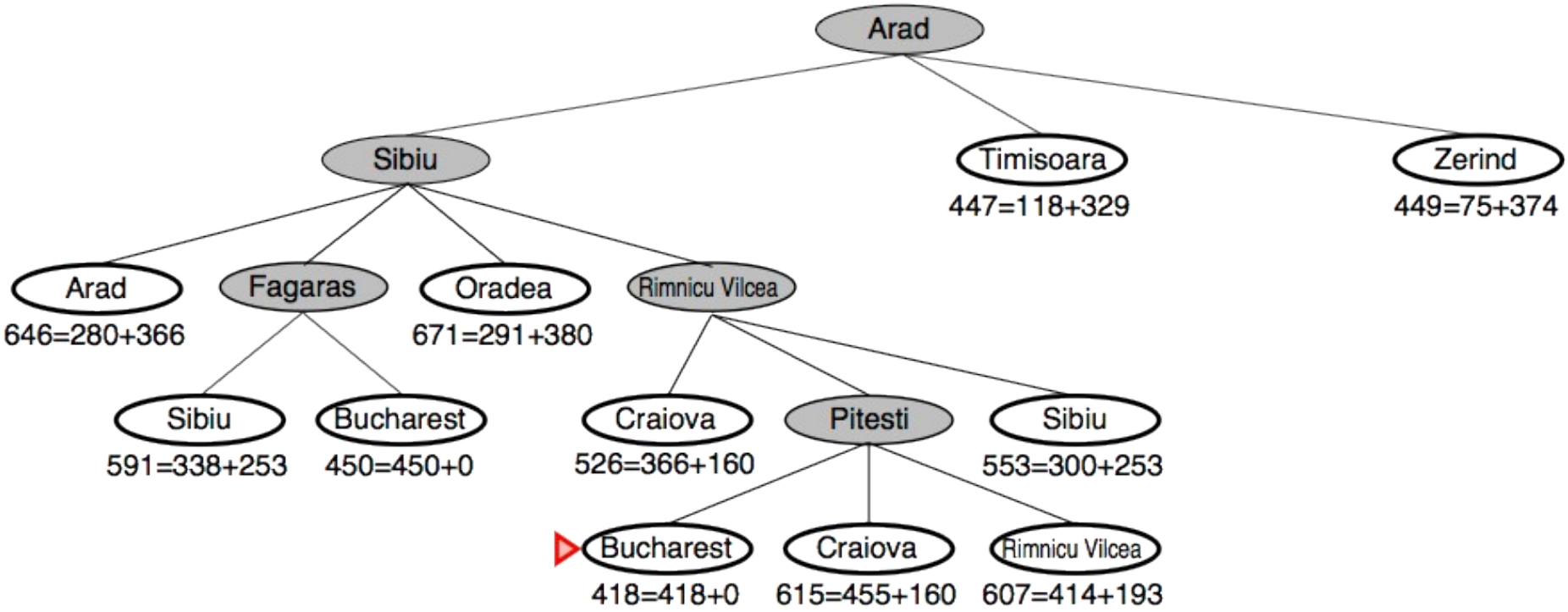
Universidad Zaragoza

Ejemplo Rumanía: A* y h_{SLD}

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



Ejemplo Rumanía: A* y h_{SLD}

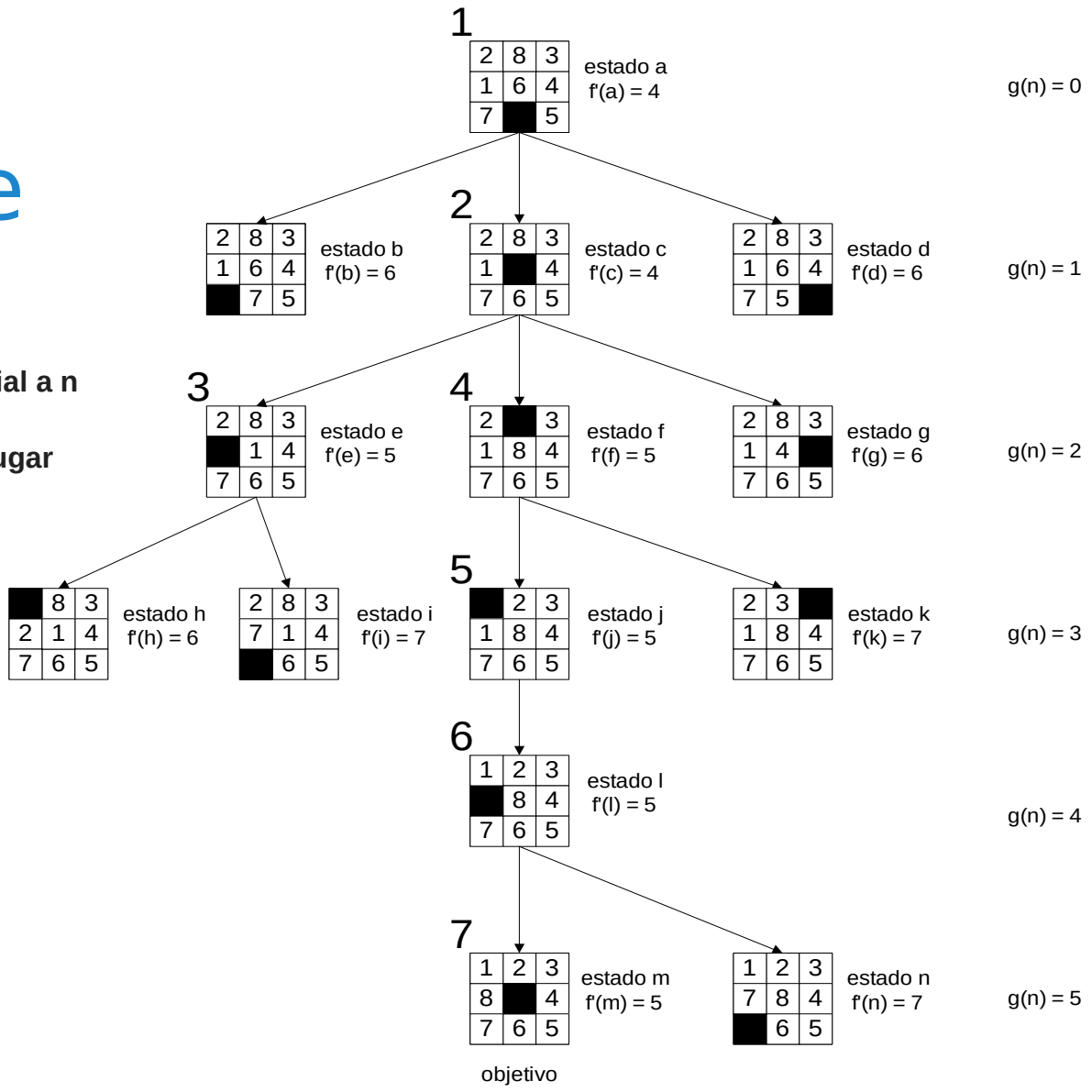


- **A* en árbol: Solución óptima si h(n) es admisible**

Universidad Zaragoza

A*/8-puzzle

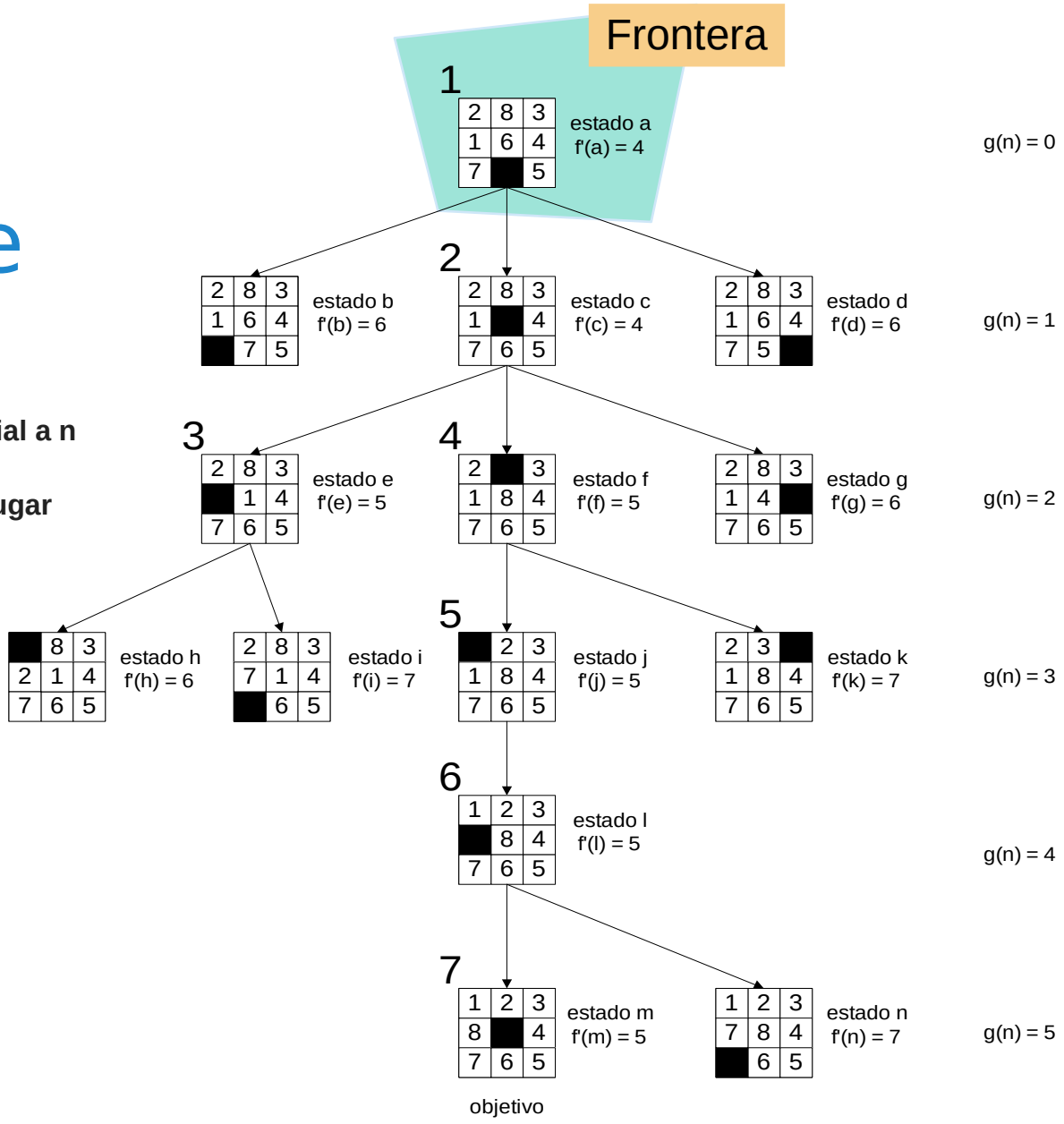
- **f'(n) = g(n) + h'(n)**
- **g(n) = distancia real del estado inicial a n**
- **h'(n) = número de piezas fuera de lugar**



Universidad Zaragoza

A*/8-puzzle

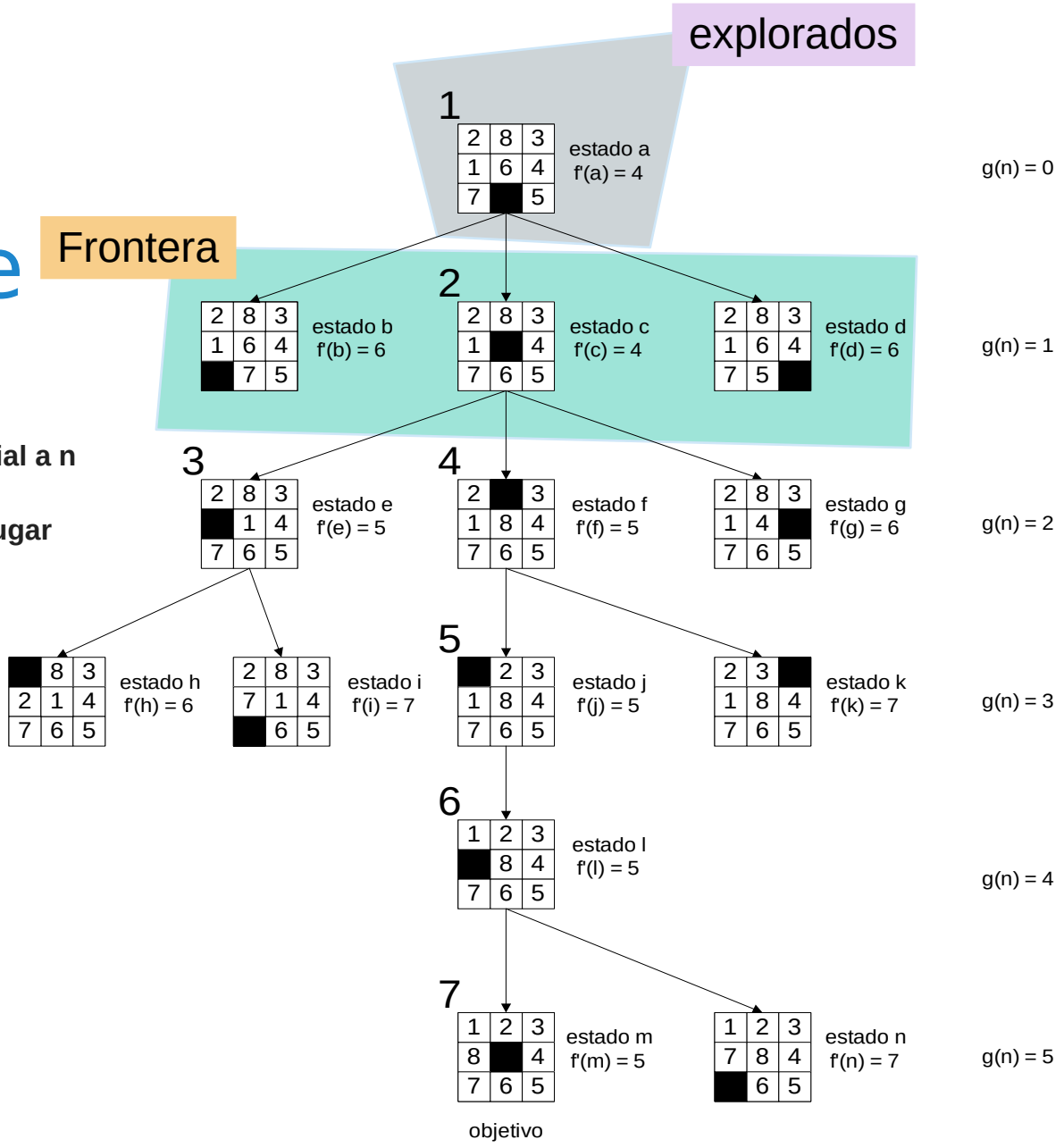
- **f'(n) = g(n) + h'(n)**
- **g(n) = distancia real del estado inicial a n**
- **h'(n) = número de piezas fuera de lugar**



Universidad Zaragoza

A*/8-puzzle

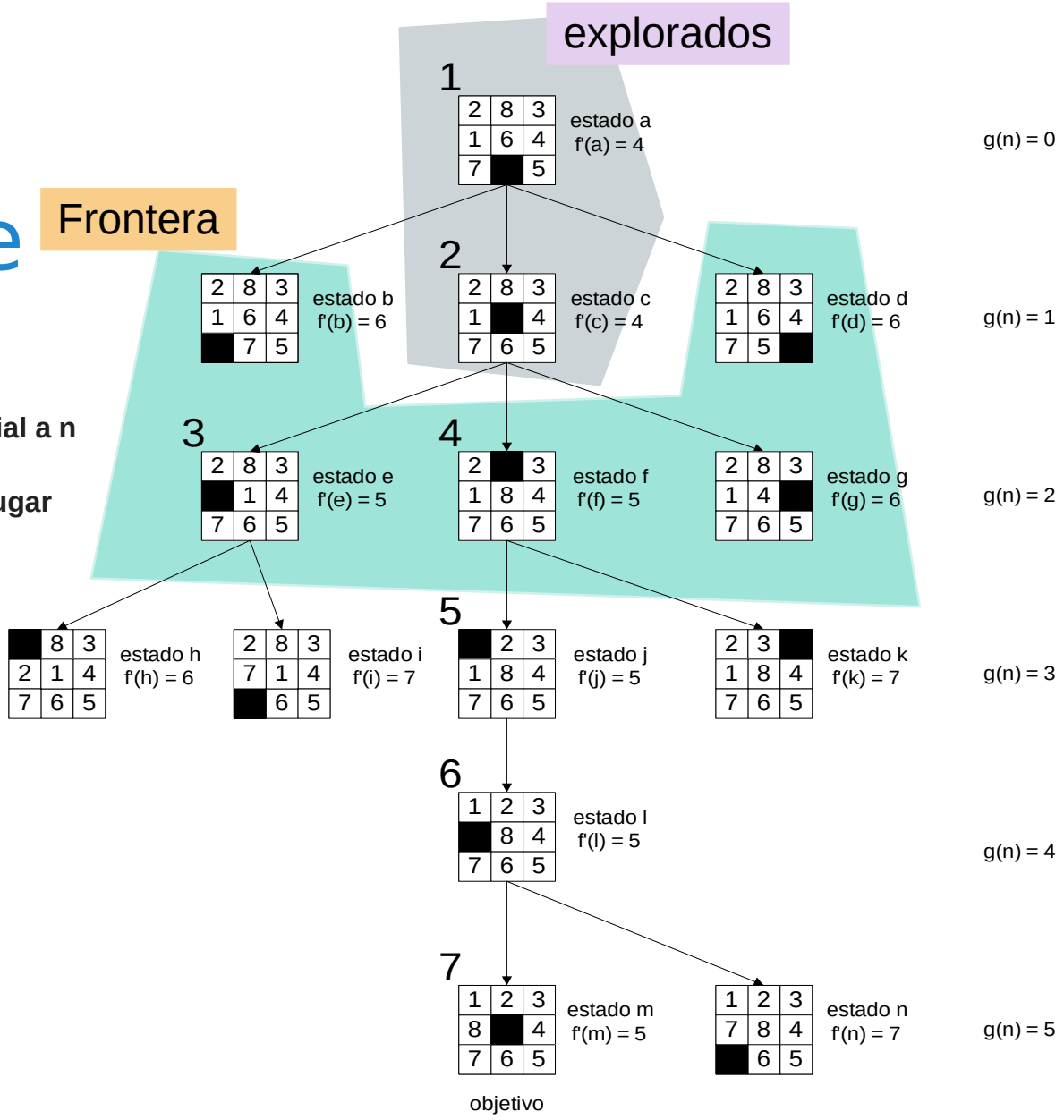
- **f'(n) = g(n) + h'(n)**
- **g(n) = distancia real del estado inicial a n**
- **h'(n) = número de piezas fuera de lugar**



Universidad Zaragoza

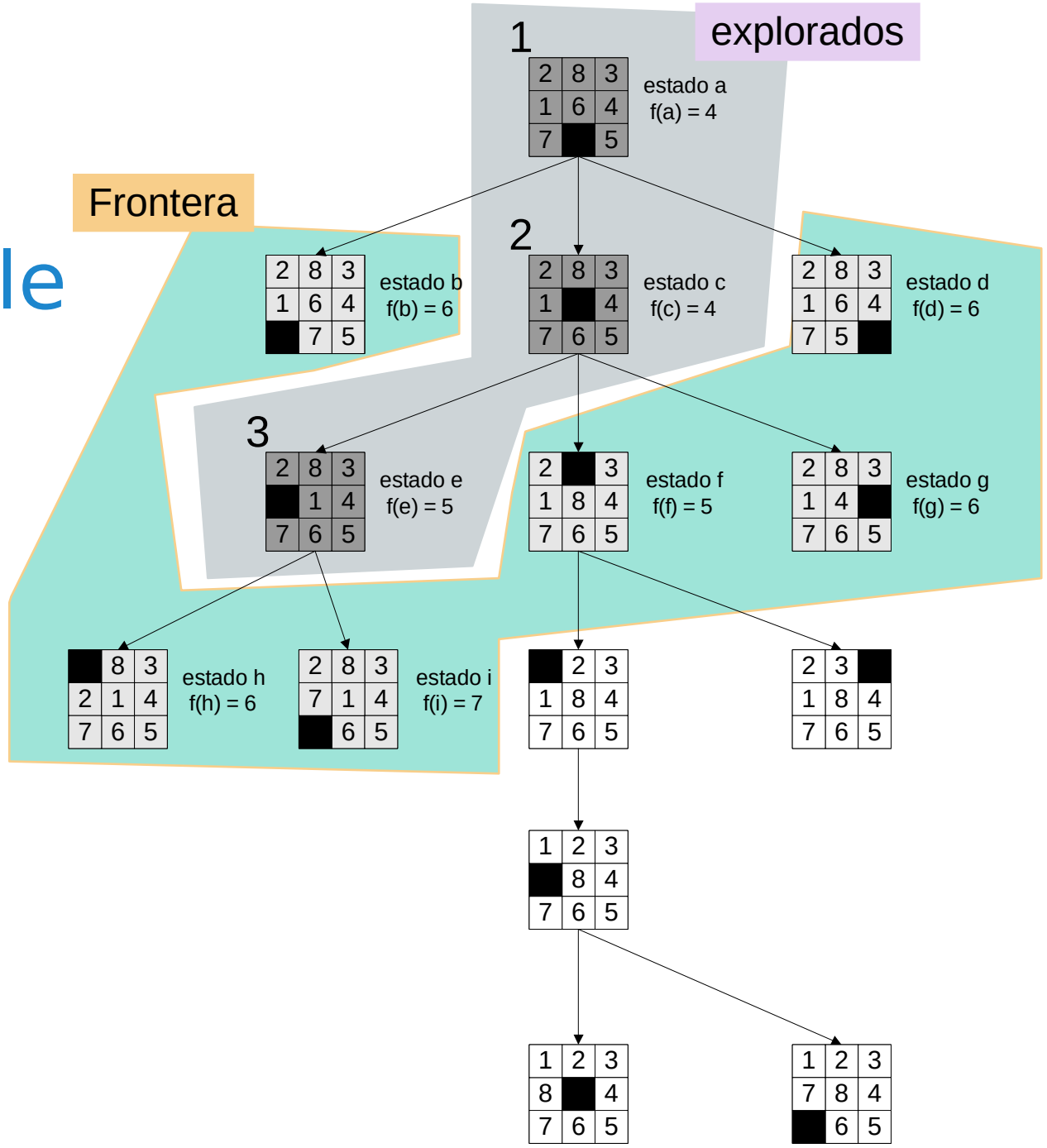
A*/8-puzzle

- **f'(n) = g(n) + h'(n)**
- **g(n) = distancia real del estado inicial a n**
- **h'(n) = número de piezas fuera de lugar**



Universidad Zaragoza

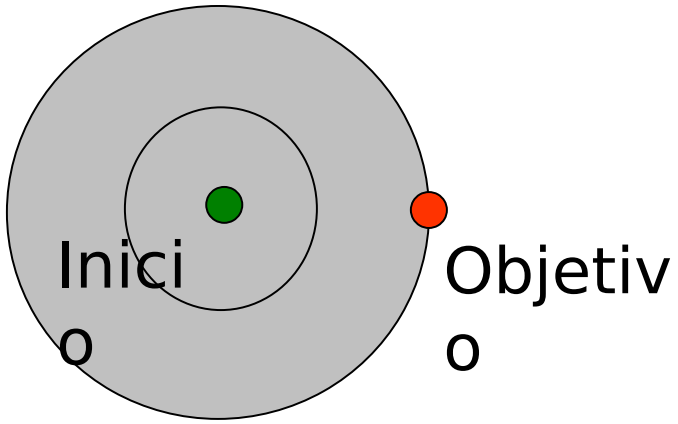
A*/8-puzzle

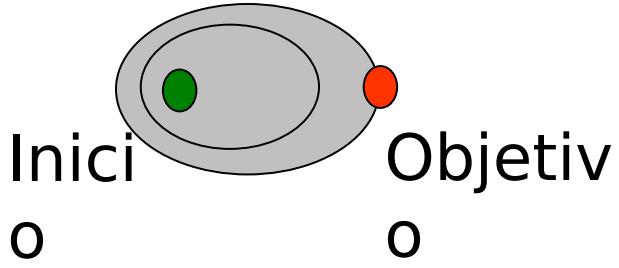


Universidad Zaragoza

Efecto de la función heurística

Guía la búsqueda **hacia el objetivo** en lugar de mirar **todas las posibilidades alrededor**

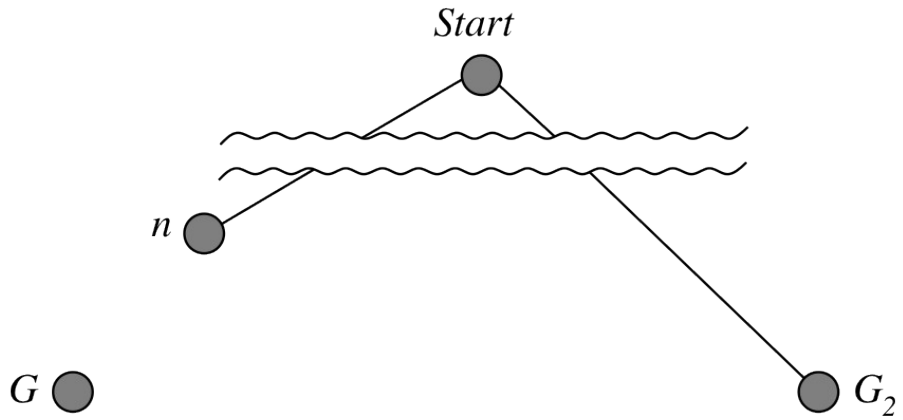




Búsqueda ciega

Informada

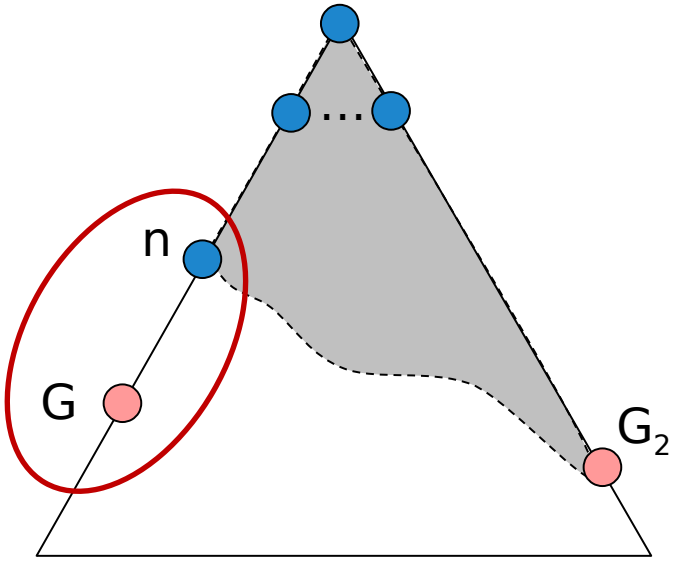
Optimalidad de A* **en árbol**



- **Supongamos**
  - **G** es un objetivo óptimo
  - ***G_2*** es un objetivo subóptimo
  - **h** es admisible.
  - **n** es un nodo en el camino óptimo a G
- **Queremos demostrar que G** se expandirá antes que G_2. Es decir, si h es admisible, la búsqueda A* en árbol es óptima.

Optimalidad de A* **en árbol**

- **Prueba**
  - Supongamos que G_2 está en la frontera
  - ***Algún nodo ancestro de n*** *estará en la frontera también (**Puede ser el propio G**)*
  - Podemos demostrar que **n** (un nodo camino del óptimo)se expandirá antes que **G_2**
    1. **f(n) es menor o igual que f(G)**



n es un nodo camino del óptimo, g(n) es por lo tanto el coste óptimo hasta n, y h(n) subestima => f(n)=g(n)+ h(n) <=g(G)

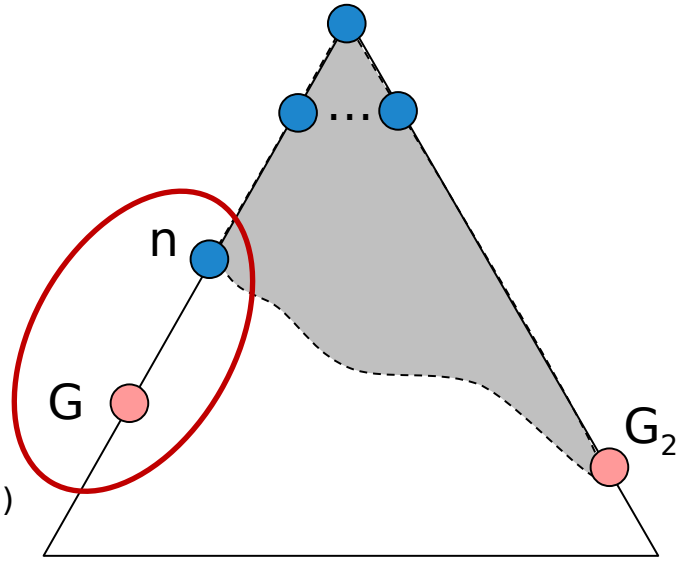
f(n) = g(n) + h(n) Definición f(n)

f(n) <= g(G) **h es admisible**

g(G) =f(G) ⇒ f(n)<=f(G) h = 0 en objetivo

Optimalidad de A* **en árbol**

- **Prueba**
  - Supongamos que G_2 está en la frontera
  - ***Algún nodo ancestro de n*** *estará en la frontera también (**Puede ser el propio G**)*
  - Podemos demostrar que **n** se expandirá antes que **G_2**
    1. **f(n) es menor o igual que f(G)**
    2. **f(G) es menor que f(G_2)**



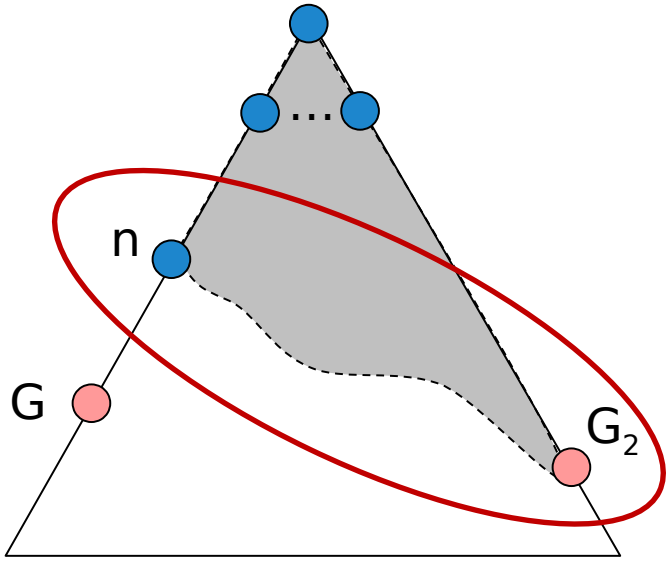
➡ f(n)<=f(G)

g(G) < g(G_2) **G_2 es suboptimo**

f(G) < f(G_2) h = 0 en objetivo

Optimalidad de A* **en árbol**

- **Prueba**
  - Supongamos que G_2 está en la frontera
  - ***Algún nodo ancestro de n*** *estará en la frontera también (****Puede ser el propio G****)*
  - Podemos demostrar que **n** se expandirá antes que **G_2**
    1. **f(n) es menor o igual que f(G)** ➡ f(n)<=f(G)
    2. **f(G) es menor que $f(G_2)$** ➡ $f(G)<=f(G_2)$
    3. **n se expande antes que G_2**
  - **Todos los ancestros de G se expanden antes que G_2**
  - **G se expande antes que G_2**
  - **A* es óptima**



$f(n) \leq f(G) \leq f(G_2)$

Pero … la búsqueda en grafo

- Descarta nuevos caminos a los estados repetidos
  - La prueba anterior no es válida

Universidad Zaragoza
1542

Descripción informal de la búsqueda en grafo

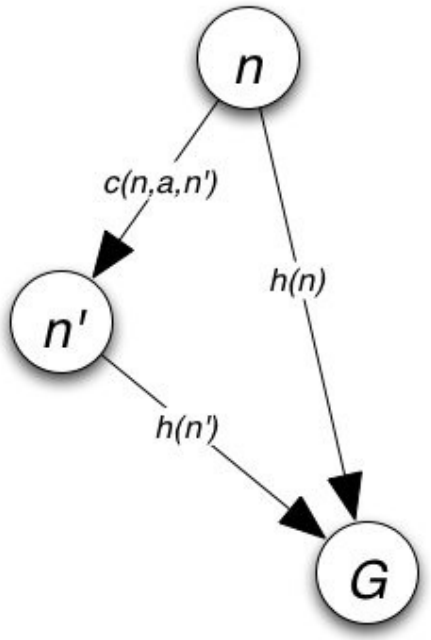
function BÚSQUEDA-GRAFO(***problema***, ***estrategia***) **returns** **solución** o **fallo**
Inicializa la ***frontera*** utilizando el estado inicial del problema
Inicializa el conjunto de nodos **explorados** a vacío
loop do
if la ***frontera*** está vacía **then return *fallo***
elige un ***nodo*** hoja de la frontera de acuerdo a una ***estrategia***
if el ***nodo*** contiene un nodo objetivo **then return *solución***
añade el ***nodo*** al conjunto de nodos ***explorados***
expande el nodo elegido, añadiendo los nodos resultantes a la frontera
sólo si no está en la ***frontera*** o en el conjunto ***explorados***

Una estrategia se define por el orden de expansión de nodos

Pero … la búsqueda en grafo

- Descarta nuevos caminos a los estados repetidos
  - La prueba anterior no es válida
- Solución:
  - **Añadir nuevos caminos a nodos repetidos eliminando el más costoso (en los nodos de la FRONTERA)**.
  - **Asegurar que se sigue primero el camino óptimo a cualquier estado repetido**.
    - **Requisito extra sobre *h(n)***: consistencia(monotonicidad)

Consistencia



- Una heurística es consistente si

$$h(n) \leq c(n,a,n') + h(n')$$

- Si h es consistente

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\geq g(n) + h(n) \\ &\geq f(n) \end{aligned}$$

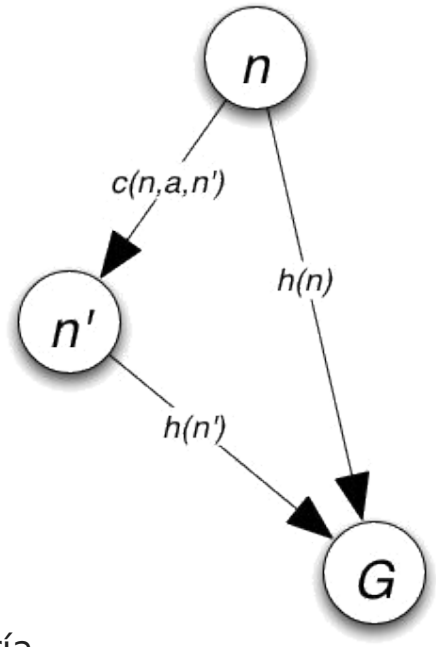
por lo tanto **f(n) es no decreciente a lo largo de cualquier camino**.

A* Búsqueda en grafo

1. f(n) es no decreciente a lo largo de cualquier camino

$$f(n') \geq f(n)$$

2. Cuando se selecciona un nodo n para expansión, el camino óptimo a este nodo ha sido encontrado
   - Si no fuera el caso, habría otro nodo n' en la frontera por el que pasaría el camino óptimo desde el origen al nodo n. **Por ser f(n) una función no decreciente, f(n') tendría menor valor** y tendría que haber sido seleccionado primero.



De las dos observaciones precedentes se sigue que ***una secuencia de nodos expandidos por A* utilizando una búsqueda en grafo es en orden no decreciente de f(n)***.

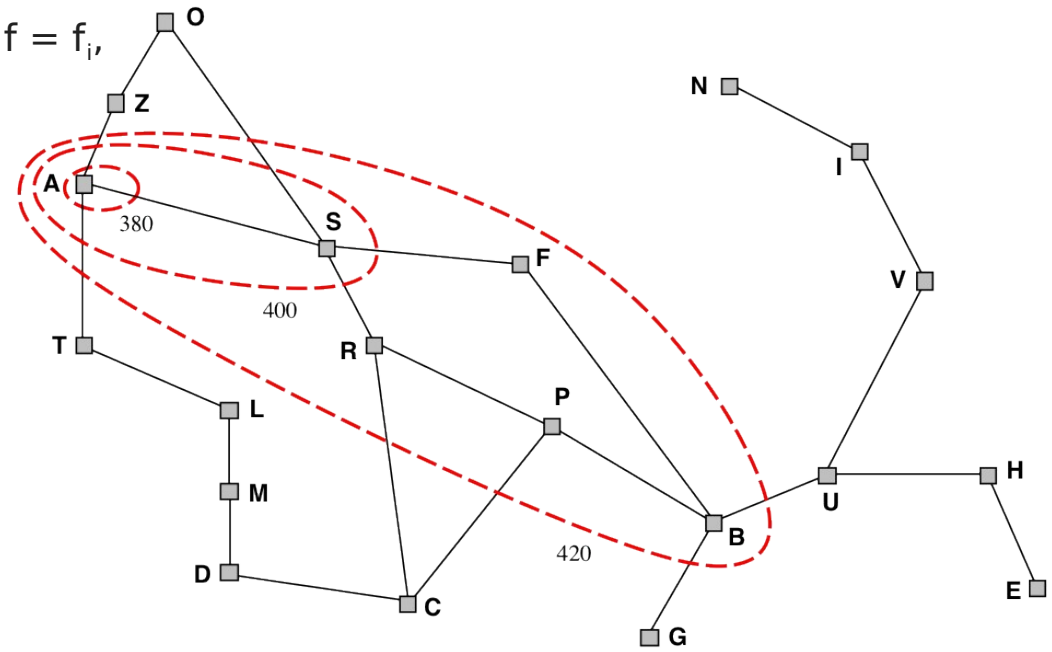
Universidad Zaragoza

Optimalidad de A* (grafo)

- A* expande nodos en orden no decreciente del valor de f
- Se pueden dibujar contornos en el espacio de estados
  - La búsqueda de coste uniforme añade círculos
  - Añade gradualmente “f-contornos” de nodos:

Contorno i tiene todos los nodos con $f = f_i$,

Donde $f_i < f_{i+1}$



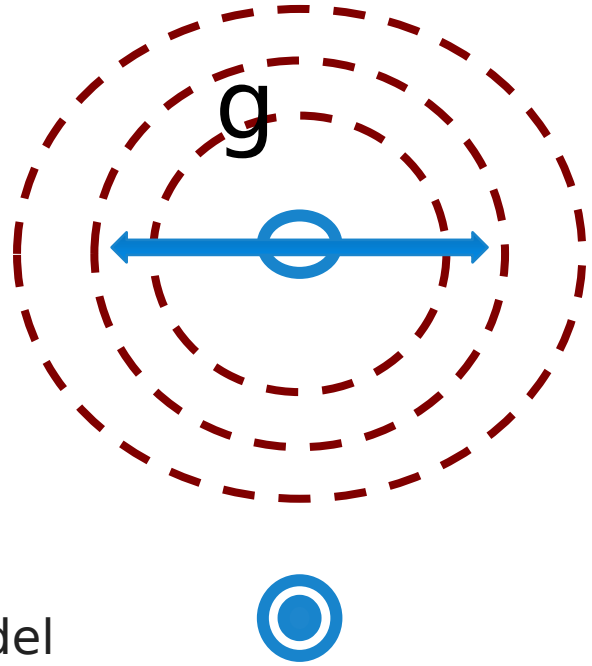
Evaluación Búsqueda A*

- Completitud: SI
  - Puesto que los f-contornos van creciendo.
    - Si C* es el coste de la solución óptima podemos decir
      - **A* Expande todos los nodos con f(n) <C***
      - A* podría expandir alguno de los nodos en el contorno objetivo (donde f(n) = C*) antes de seleccionar el nodo objetivo.
  - A no ser que haya infinitos nodos con ***f<f(G)***

Evaluación Búsqueda A*

- Completitud: Si
- Complejidad Temporal
  - El número de nodos expandidos es todavía exponencial con la longitud de la solución
    - Los detalles de este análisis están fuera del alcance del curso, …. pero veamos la idea intuitiva.

Evaluación Búsqueda A*



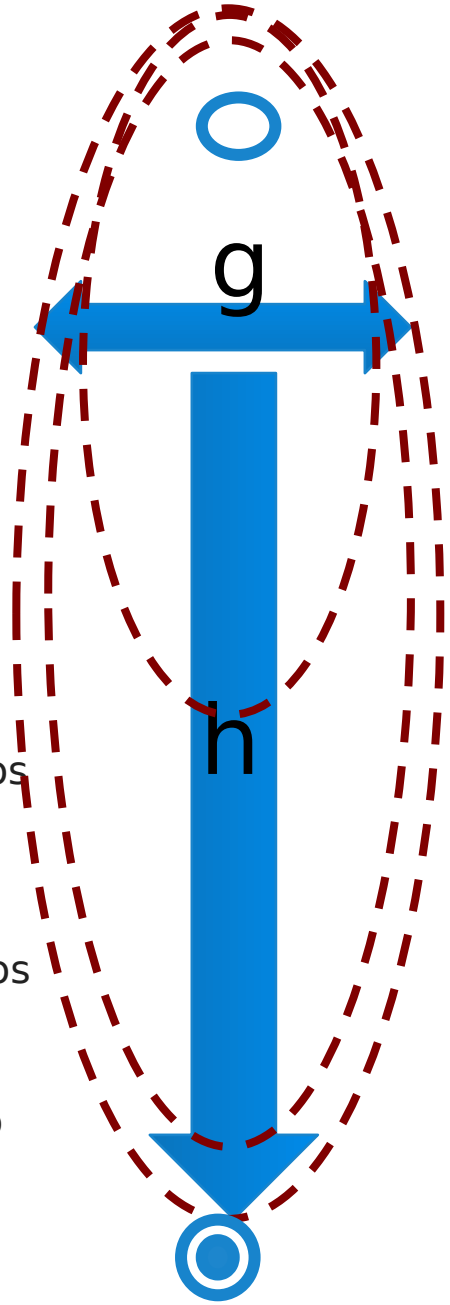
- Completitud: Si
- Complejidad Temporal
  - El número de nodos expandidos es todavía exponencial con la longitud de la solución
    - Los detalles de este análisis están fuera del alcance del curso, …. pero veamos la idea intuitiva.

Hay que considerar que el coste de este algoritmo es $O(b^d)$ en el peor caso. Si por ejemplo, la función heurística h(n) = 0, el algoritmo se comporta como una búsqueda en anchura gobernada por ***f(n) = g(n).***

Universidad Zaragoza

Evaluación Búsqueda A*

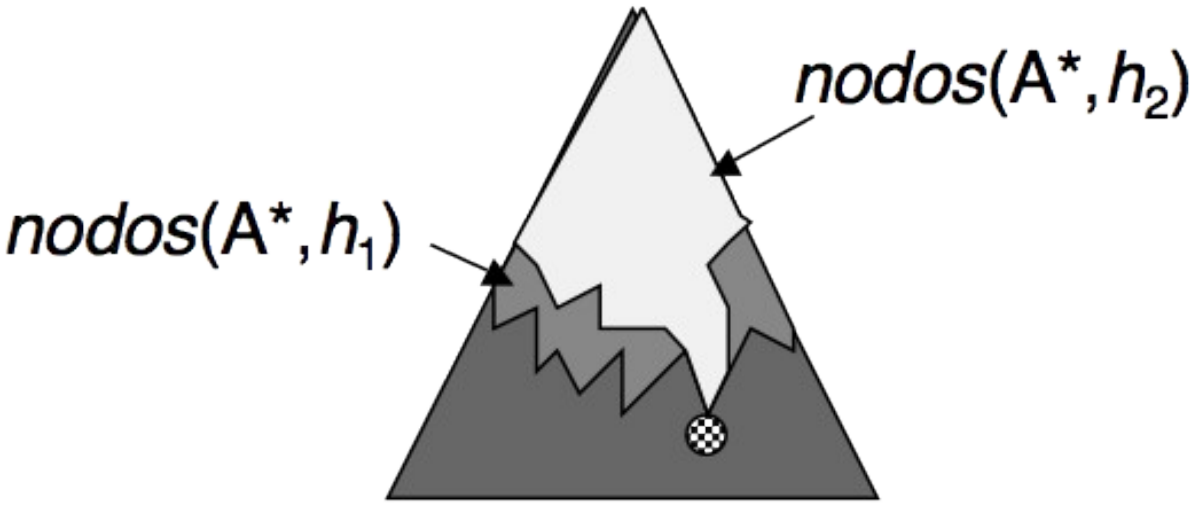
- Completitud: Si
- Complejidad Temporal
  - El número de nodos expandidos es todavía exponencial con la longitud de la solución
    - Lo que hace que este algoritmo **pueda tener coste temporal inferior** es la **bondad de la función h**. Podemos interpretar que g y h gobiernan el comportamiento en anchura o profundidad del algoritmo
      - Cuanto **más cercana al coste real sea h**, mayor será el **comportamiento en profundidad** del algoritmo, pues los nodos que están más cerca de la solución se explorarán antes
      - Cundo la **información no sea fiable**, el coste del camino explorado hará que otros menos profundos tengan coste mejor, y **se abre la búsqueda en anchura**.



Universidad Zaragoza
1542

Heurísticas más informadas

- nodos (A*, h): nodos expandidos por A* con h
- h1 y h2 admisibles,
- h2 es más informada que h1 si h2(n) > h1(n)
  ∀n nodos (A*,h2) ⊂ nodos (A*,h1)



Universidad Zaragoza
1542

Heurísticas más informadas

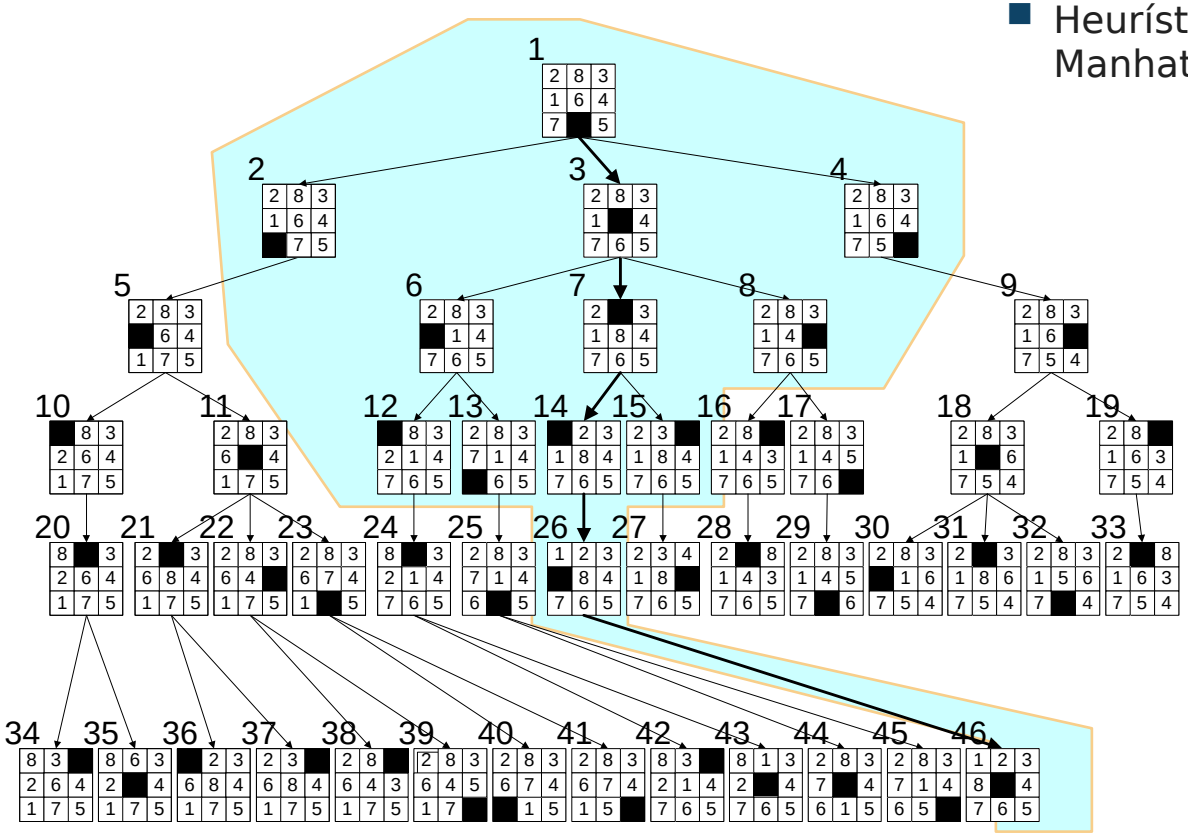
- Dadas dos heurísticas A* h_1 y h_2,
  - se dice que h_2 está ***mejor informada*** que h_1
  - si para cualquier estado n del espacio de búsqueda
  - $\boldsymbol{h_1(n) <= h_2(n)}$

- Comparaciones en el 8-puzzle
  - La búsqueda en anchura es el peor A* ($h'(n) = 0$)
  - La heurística suma de piezas fuera de lugar es un A*
  - La heurística con la distancia Manhattan es A* y está mejor informada que la anterior

Universidad Zaragoza

Comparación A* y anchura en 8-puzzle

- Comparación del espacio de estados entre los algoritmos A* y primero en anchura
  - Heurística utilizada: distancia Manhattan



Evaluación Búsqueda A*

- Completitud: Si
- Complejidad temporal:
  - Si la heurística no es buena, es como una búsqueda en anchura
  - Exponencial con la longitud del camino
- Complejidad espacial:
  - Mantiene todos los nodos generados en memoria
  - **La memoria es el mayor problema, no el tiempo**.

Evaluación Búsqueda A*

- Completitud: SI
- Complejidad temporal: Exponencial con la longitud del camino
- Complejidad Espacial: Se almacenan todos los nodos
- Optimalidad: SI
  - No se expande f_{i+1} hasta que f_i se finaliza.
  - A* expande todos los nodos con $f(n) < C^*$
  - A* expande algunos nodos con $f(n) = C^*$
  - A* No expande nodos con $f(n) > C^*$

Universidad Zaragoza
1542

Medida calidad de la heurísticas

$$\textbf{Factor de ramificación medio real} = \frac{\textbf{total-estados-generados}}{\textbf{nodos-expandidos}}$$

$$\textbf{Factor ramificación medio árbol generado} = \frac{\textbf{total-nodos-generados}}{\textbf{nodos-expandidos}}$$

Universidad Zaragoza

Calidad de la heurística/rendimiento

- **Factor de ramificación efectivo b***
- **Mide la calidad de la heurística. Si el A* ha generado N nodos**
  - Es el factor de ramificación que un árbol uniforme de profundidad d tendría para contener N+1 nodos

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

  - Puede variar entre las instancias de problemas, pero es suficientemente constante para problemas difíciles
    - Es una buena medida de la utilidad de la heurística.
    - Un buen valor de b* es 1.
    - Ejemplo, si d=5 y N= 52, b*= 1.92

Universidad Zaragoza

Calculo del factor de ramificación efectivo

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

$$\begin{array}{r} S_n=1+b^*+(b^*)^2+\ldots+(b^*)^d \\ -\quad b^*S_n=b^*+(b^*)^2+\ldots+(b^*)^d+(b^*)^{d+1} \\ \hline S_n(1-b^*)=1-(b^*)^{d+1} \end{array}$$

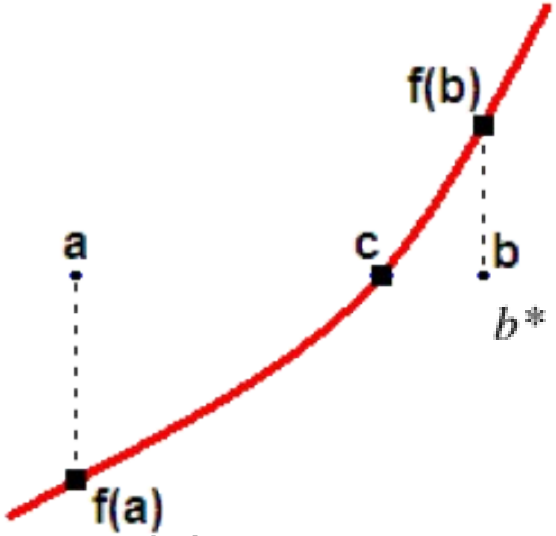
$$N+1=\frac{1-(b^*)^{d+1}}{1-b^*} \rightarrow 0=\frac{1-(b^*)^{d+1}}{1-b^*}-1-N \rightarrow$$

$$\rightarrow 0=\frac{1-(b^*)^{d+1}-(1-b^*)}{1-b^*}-N \rightarrow 0=\frac{b^*(1-(b^*)^d)}{1-b^*}-N$$

Universidad Zaragoza

$$0 = \frac{b^*(1-(b^*)^d)}{1-b^*} - N$$

Resolver la ecuación por Bolzano



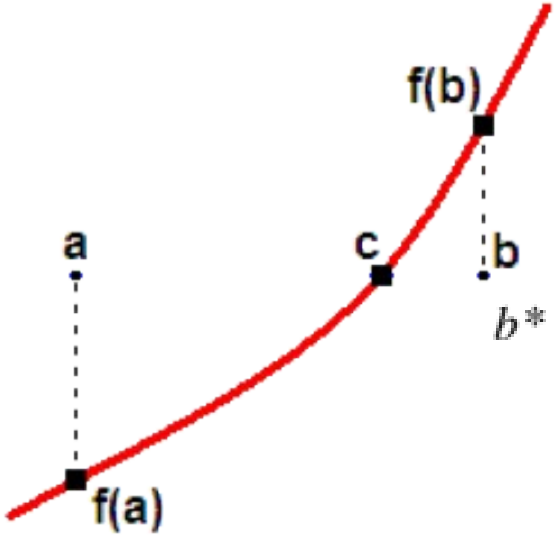
- Sea **f(x) continua en [a,b],** con **f(a)·f(b) < 0** (signos distintos)
- Entonces EXISTE AL MENOS un c∈[a,b] t.q. f(c)=0

Método númerico de resolución

- Paso 1: Estableceremos intervalos [a,b] que contengan una única raíz de f(x) (Por ejemplo, en el 8 puzzle [1,4])
- Paso 2: Utilizaremos algún método numérico iterativo de forma que $\mathbf{x_k}$ será un aproximación a la solución exacta tras **k** iteraciones.

Universidad Zaragoza

$$0 = \frac{b^*(1-(b^*)^d)}{1-b^*} - N$$



Método bibisección

Llamemos $[a_0,b_0]$ a nuestro intervalo [a,b]

El error al tomar el punto medio como solución es ***error*** ≤ (b-a)/2

MIENTRAS QUE error > error_admitido

Llamemos **m** al punto medio de **a0** y **b0** =**m**=$(b_0-a_0)/2$

Si f(m)=0, casualmente hemos encontrado: SOL = m. PARAMOS

Si f(m)≠0 elegimos entre

$[a_0,m]$ si es que $f(a_0)\cdot f(m)<0$, haciendo $[a_1,b_1]$= [a0,m]

$[m,b_0]$ si es que $f(m)\cdot f(b_0)<0$, haciendo [a1,b1]= [m,b0]

Calculo de nuevo el error con el nuevo intervalo

FIN MIENTRAS QUE

SOL=punto medio el ultimo intervalo considerado

Universidad Zaragoza

Rendimientos en 8-puzle

- 1200 problemas aleatorios con soluciones de longitudes entre 2 y 24.
  - Los datos se promedian sobre 100 instancias del 8-puzzle para cada longitud de la solución d.

- Profundización iterativa
- A* con heurística **a** (número de piezas en **posición errónea**)
- A* con heurística **b**(**distancia Manhattan**)

	Coste de la búsqueda Nodos Generados			**Factor ramific. efectivo**		
d	IDS	A*(h_a)	A*(h_b)	IDS	A*(h_a)	A*(h_b)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3056	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.27
22	-	18094	1219	-	1.48	1.28
24	-	39135	1641	-	1.48	1.26

Generar heurísticas admisibles relajando el problema

- Las soluciones admisibles pueden ser derivadas mediante una versión relajada del problema
  - Versión relajada del 8-puzle h_1 *(Casillas descolocadas)* **: Una casilla se puede mover a cualquier sitio**, en vez de a la casilla adyacente.

    En este caso, $h_1(n)$ da la solución más corta.
  - Versión relajada del 8-puzle para h_2 (distancia de Manhatan): una casilla **puede moverse a cualquier cuadro adyacente, incluso aunque esté ocupada**. En este caso , $h_2(n)$ *da la solución más corta.*

Universidad Zaragoza
1542

Generar heurísticas admisibles relajando el problema

- La solución óptima de un problema relajado subestima la solución óptima del problema real.

- Un problema relajado añade “caminos” entre estados en el problema relajado[1].
  - **Cualquier solución en el problema real es solución en el problema relajado. Pero el problema relajado puede tener mejores soluciones utilizando atajos**.
  - El coste de una solución óptima en un problema relajado es una heurística del problema original.

- Si generamos varias heurística y no tenemos claramente una mejor, podemos utilizar **$h(n) = max\{h_1(n), \ldots h_m(n)\}$**

1Un programa denominado ABSOLVER puede generar automáticamente heurísticas a partir de la definición del problema, utilizando esta técnica.

A* (Mi memoria se acaba)

- El A* tiene limitaciones de espacio
  - Puede acabar degenerando en una búsqueda en anchura si h no es buena
  - Problemas con soluciones a mucha profundidad o tamaño de espacio de estados grandes
  - Búsquemos alternativas con menos necesidades de espacio.

Universidad Zaragoza

Busqueda heurística con memoria acotada

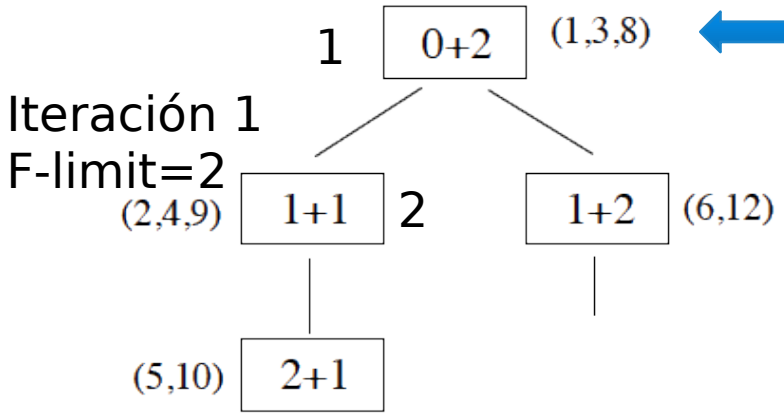
- Solución a los problemas de memoria del A* (manteniendo completitud y optimalidad)
  - Iterative-deepening A* (**IDA***)
    - Igual que IDS, **búsqueda en profundidad**, pero **con límite dado por f en cada iteración**. (Sólo usa **f-limit en curso**)
    - Imponemos un **límite** a ***f*- (*g*+*h*)** en lugar de la profundidad (límite: menor valor de f de la iteración anterior).
  - Recursive best-first search(**RBFS**)
    - Algoritmo recursivo que intenta imitar la búsqueda primero el mejor con espacio lineal.
    - En lugar de continuar indefinidamente en una rama, usa una variable **f-limit**, que es **la mejor alternativa**
  - (simple) Memory-bounded A* ((**S**)**MA***)
    - Elimina el nodo con peor coste cuando la memoria está llena.

IDA*

- Sufre las mismas dificultades que la **Iterative deepening search** (idéntico, pero con f=g+h, en lugar de la profundidad)
  - Si se genera un nodo cuyo coste excede el límite en curso se descarta. Para cada nueva iteración, **el límite se establece en el del mínimo coste de cualquier nodo descartado en la iteración anterior**.
- El algoritmo expande nodos en orden creciente de coste, por lo que el primer objetivo encontrado es óptimo.
- *Evita la sobrecarga de mantener ordenada la cola de nodos con nodos más allá del límite*.
- IDA* en el peor caso, h=0, **$O(b^d)$**

Universidad Zaragoza

Ejemplo ejecución IDA*

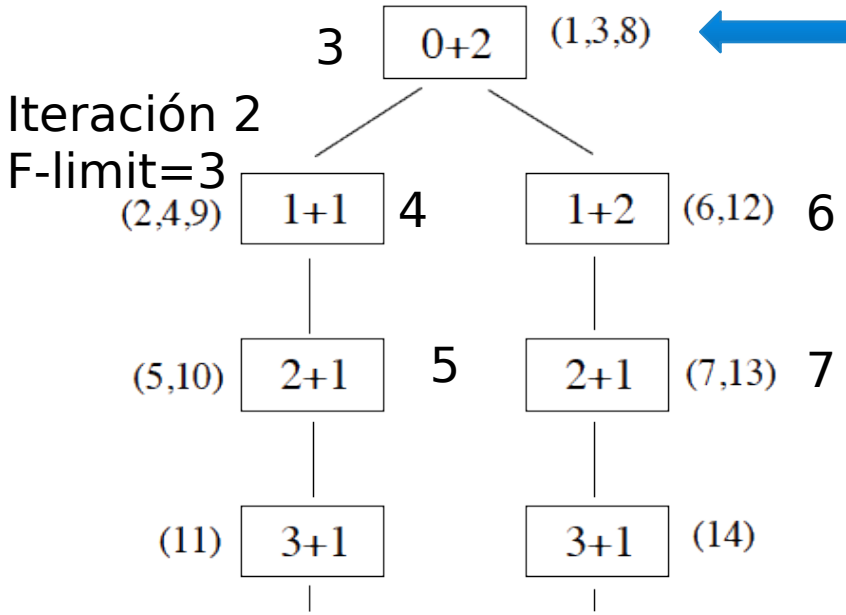


Orden de expansión

g+h

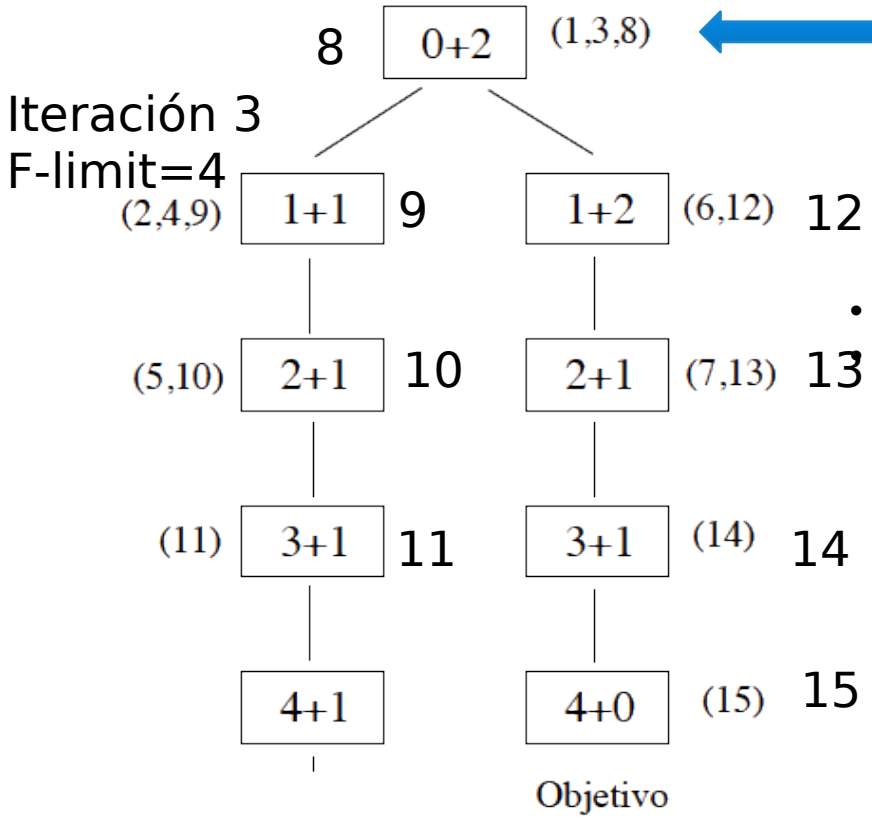
- Se puede aumentar el coste de uno en uno
- Se puede obtener el coste mas pequeño de los nodos no expandidos y usarlo en siguiente iteración

Ejemplo ejecución IDA*



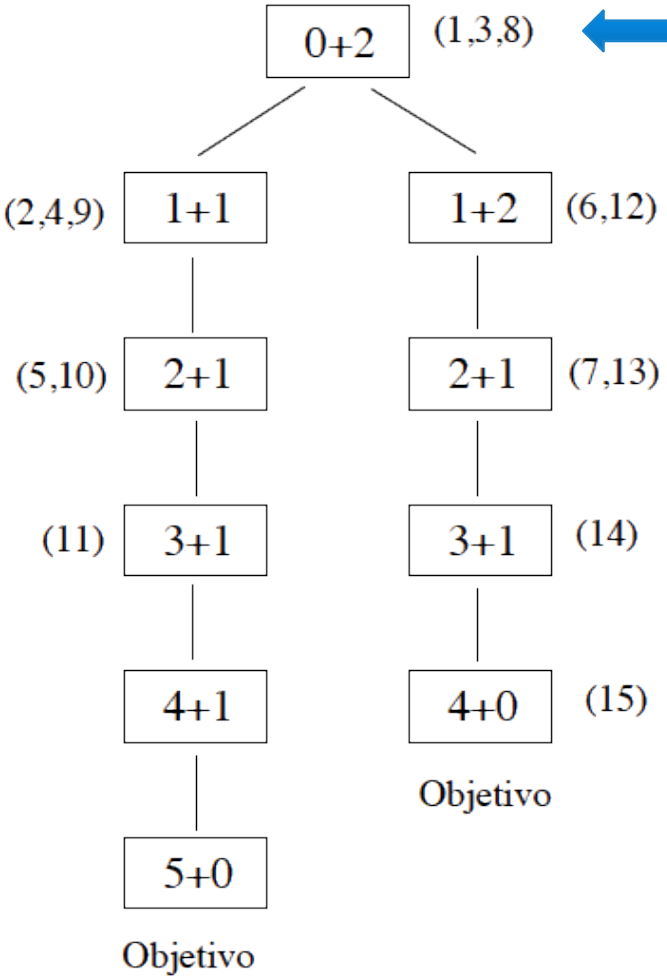
- Se puede aumentar el coste de uno en uno
- Se puede obtener el coste mas pequeño de los nodos no expandidos y usarlo en siguiente iteración

Ejemplo ejecución IDA*



- Se puede aumentar el coste de uno en uno
  Se puede obtener el coste mas pequeño de los nodos no expandidos y usarlo en siguiente iteración

Ejemplo ejecución IDA*



Orden de expansión

- Se puede aumentar el coste de uno en uno
- Se puede obtener el coste mas pequeño de los nodos no expandidos y usarlo en siguiente iteración

Universidad Zaragoza
1542

Búsqueda primero el mejor recursivo (Recursive best-first search RBFS)

- **Mantiene el valor f de la mejor alternativa** disponible de cualquier ancestro del nodo.
  - Si el valor f en curso excede el valor alternativo se vuelve al camino alternativo.
  - Al hacer **backtracking** se cambia f de cada nodo en el camino por el mejor valor de f en sus hijos.
  - **RBFS recuerda el mejor valor de f de las hojas de sub-árboles olvidados**, y puede decidir **si merece la pena volver a re-expandirlos**.
- **Algo más eficiente que IDA***, pero puede soportar una regeneración de nodos excesiva.

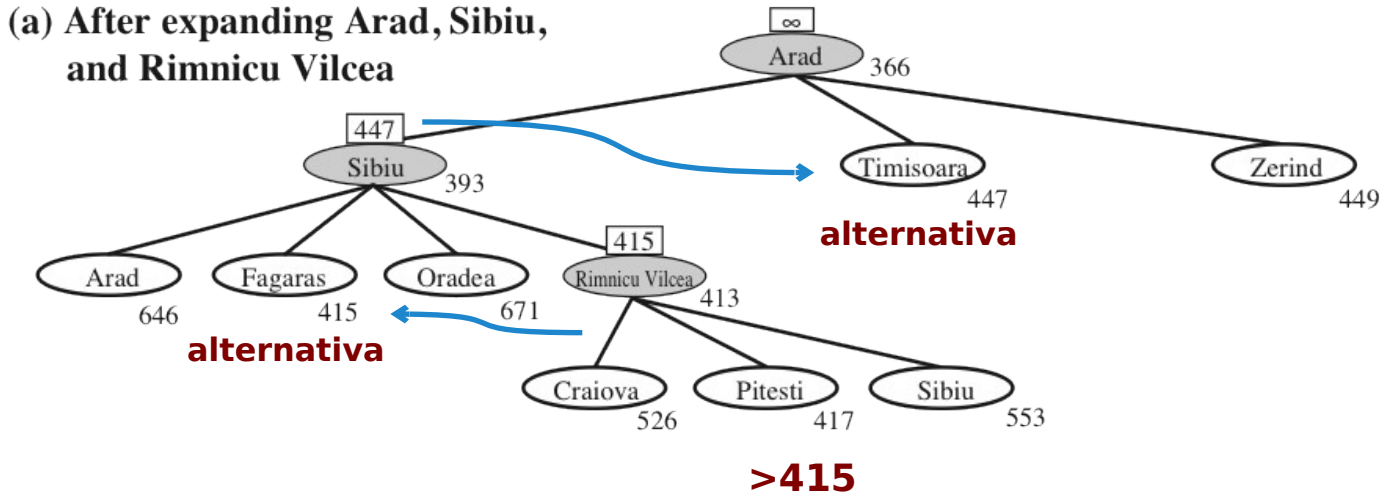
Universidad Zaragoza

Recursive best-first search

function RECURSIVE-BEST-FIRST-SEARCH(*problema*) **return** a solution or failure
    return RFBS(*problema*,MAKE-NODE(*problema.*INITIAL-STATE),∞)

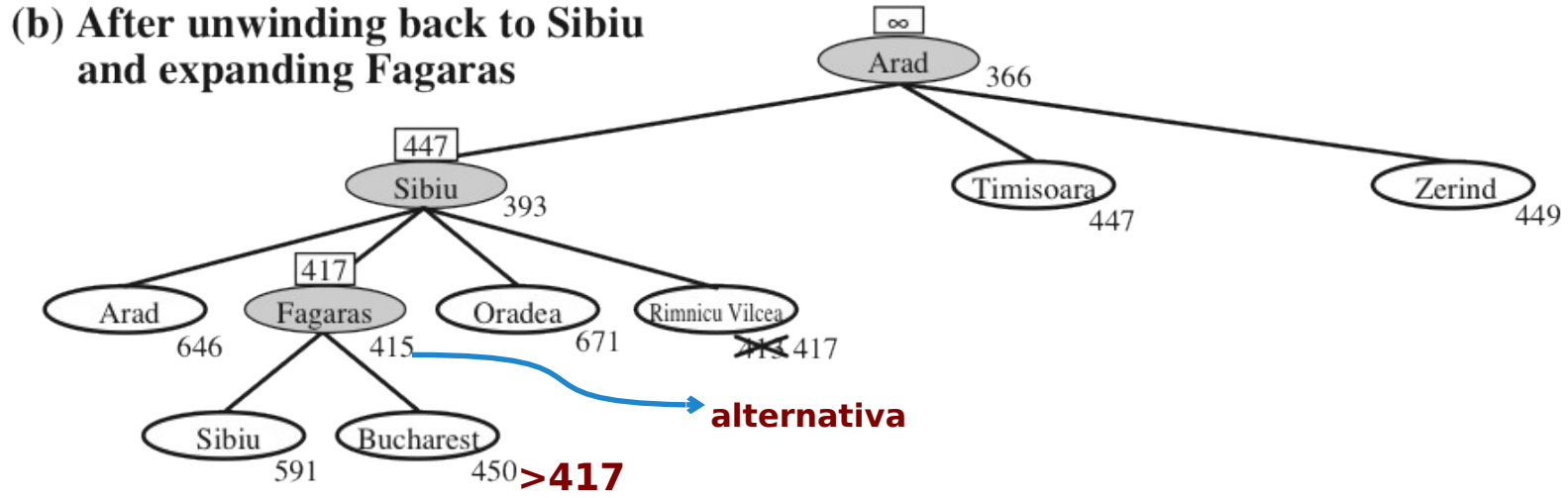
function RFBS(*problema, nodo, f_limit*) **return** solución o fallo y un nuevo límite *f-cost*
    if *problema.*GOAL-TEST(*nodo.*STATE) **then return** *nodo*
    successores ← EXPAND(*nodo, problema*)
    if *sucessores* está vacio devuelve fallo, ∞
    for each *s* **in** *sucesores* **do** **/* actualiza valores con búsqueda previa */**
        s.f ← max(*s.g* + *s.h*, *nodo.f*)
    repeat
        best* ← el menor *f*-value de los nodos en *sucesores
        if *best.f* > *f_limit* then return fallo, *best.f*
        alternativa← el segundo mejor *f*-value entre *sucesores*
        resultado, *best.f* ← RBFS*(problema, best,* min(*f_limit, alternativa*))
        if *resultado* ≠ fallo **then return** *resultado*

Recursive best-first search, ej.



- Camino hasta Rumnicu Vilcea expandido
- Sobre el nodo *f*-limit para cada llamada recursiva.
- Debajo del nodo: *f(n)*
- El camino es seguido hasta Pitesti que tiene un valor f peor que el f-limit.

Recursive best-first search, ej.

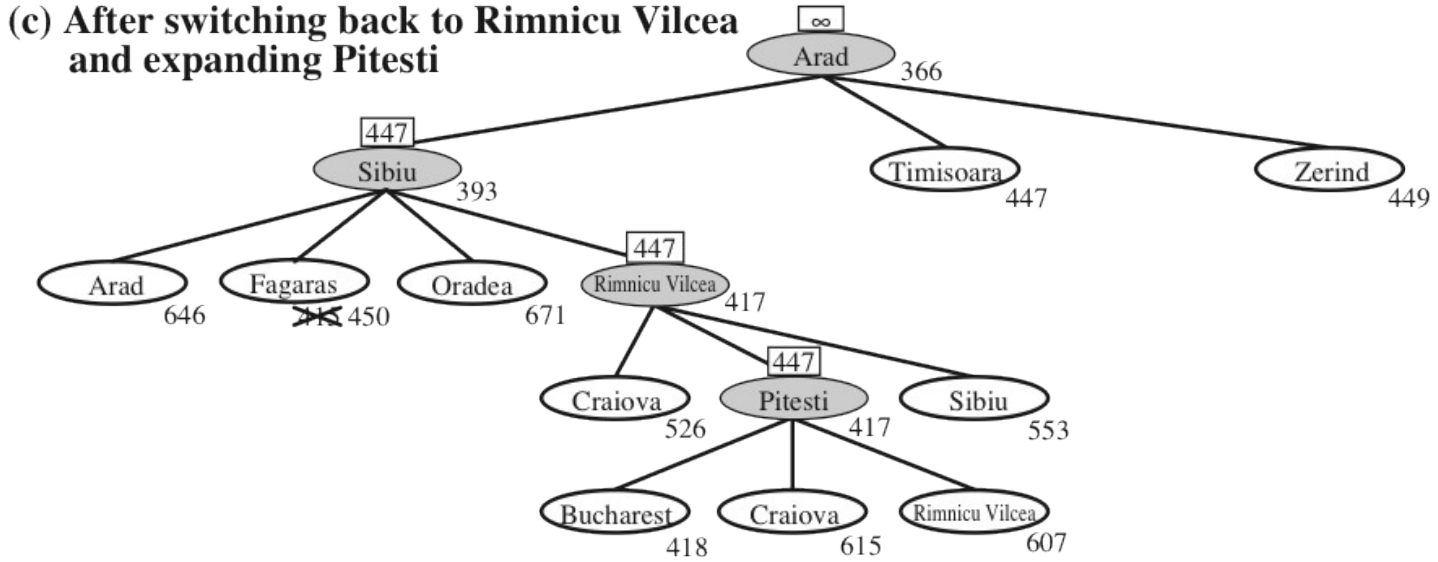


- Vuelve de llamada recursiva y almacena el mejor valor f de la hoja en curso Pitesti (417)

  result, *f* [*best*] ← RBFS(*problem*, *best*, min(***f_limit***, ***alternative***))

- *El mejor ahora es* Fagaras. Se llamada RBFS con el nuevo mejor
  - *El mejor ahora es* 450

Recursive best-first search, ej.



- Vuelve de la llamada recursiva y almacena el mejor f de la hoja en curso Fagaras

  result, *f* [*best*] ← RBFS(*problem*, *best*, min(*f_limit*, *alternative*))

- *El mejor ahora es* Rimnicu Viclea (**de nuevo**). Llama RBFS para el nuevo mejor
  - El sub-árbol es de nuevo expandido.
  - Mejor subárbol alternativo es ahora Timisoara..
- Solución encontrada es encontrada por qué 447 > 417.

Evaluación RBFS

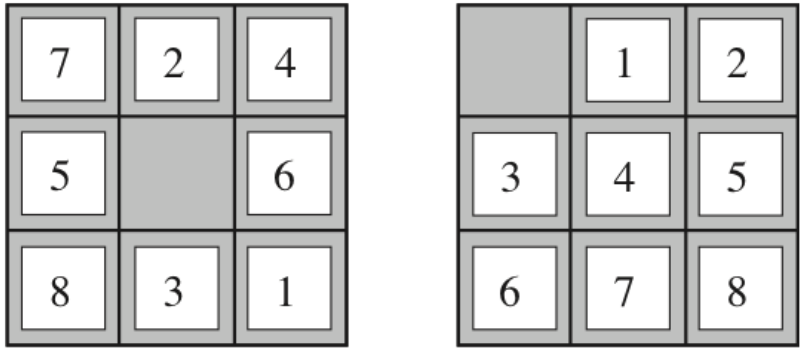
- RBFS es un poco más eficiente que IDA*
  - Todavía genera excesivos nodos
- Como A*, optima si *h(n)* es admisible
- Complejidad espacial *O(bd)*.
- Complejidad temporal difícil de caracterizar
  - Depende de la precisión de h(n) y con que frecuencia cambiamos el mejor camino.
- IDA* y RBFS utilizan **demasiada poca** memoria.

Universidad Zaragoza
1542

(simplified) memory-bounded A* (S)MA*

- Utiliza la memoria disponible.
  - Expande la mejor hoja hasta que se llena la memoria
  - Cuando se llena, SMA* elimina el peor nodo (valor de *mayor*)
  - Como RFBS devuelve el valor del nodo olvidado a su padre.
- SMA* es completa si las solución es alcanzable, óptimo si la solución es alcanzable.

Funciones heurísticas

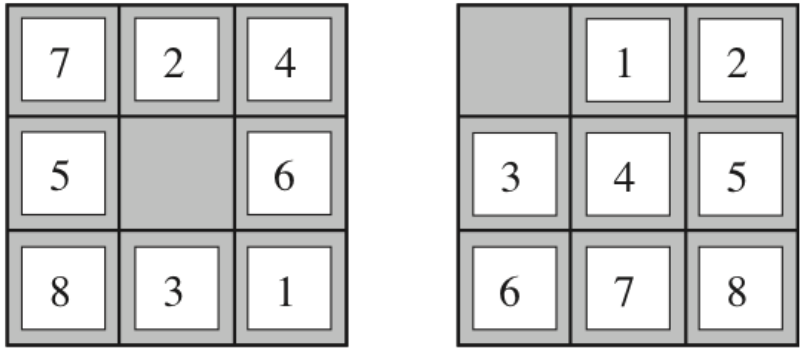


Estado inicial **Estado objetivo**

- Para el 8-puzle
  - **Coste medio de la solución** de un estado generado aleatoriamente es unos **22 pasos** (factor de ramificación b +/- 3)
  - Una **búsqueda exhaustiva en árbol** de profundidad 22 explorará : $3^{22} = 3.1 \times 10^{10}$ estados.
  - **Una búsqueda en grafo** reduce la búsqueda en un factor 170.000 porque sólo **9!/2= 181.440** estados son alcanzables
- Para el 15-puzzle
  - Una búsqueda en grafo son 15!/2, unos **10^{13} estados**

Una buena función heurística reduce el proceso de búsqueda.

Funciones heurísticas



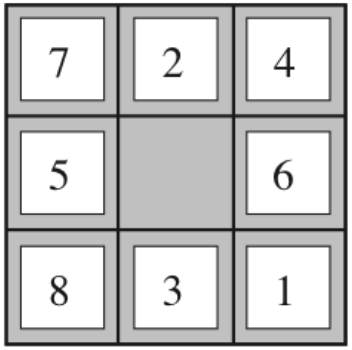
Estado inicial **Estado objetivo**

- Por ejemplo para el 8-puzle 2 heurísticas conocidas
- h_1 = número de piezas descolocadas
  - $h_1(s)=8$
- h_2 = La suma de distancias de las piezas a su posición objetivo (**distancia de manhattan**).
  - $h_2(s)=3+1+2+2+2+3+3+2=18$

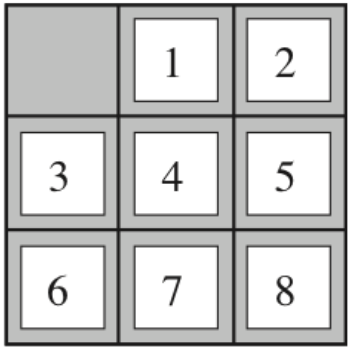
Universidad Zaragoza

Generar heurísticas: Bases de datos de patrones

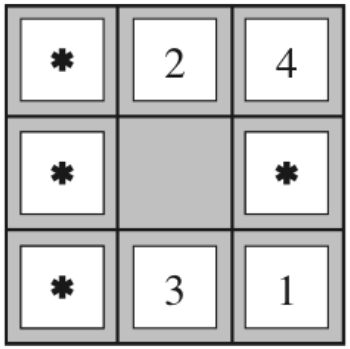
- Las soluciones admisibles pueden derivarse también de soluciones a **subproblemas** de un problema dado.
  - La idea es **almacenar el coste exacto de cada posible subproblema**.
  - Bases de datos de patrones almacenan el coste de la solución de cada posible instancia de subprobema.
  - E l coste es una cota inferior del coste real del problema.
- La heurística completa se construye usando los patrones de la BD
- **Una base de patrones se puede construir moviendo hacia atrás desde el objetivo**
- Se pueden construir varias bases de patrones y combinar $h(n) = \max\{h_1(n), \ldots h_m(n)\}$



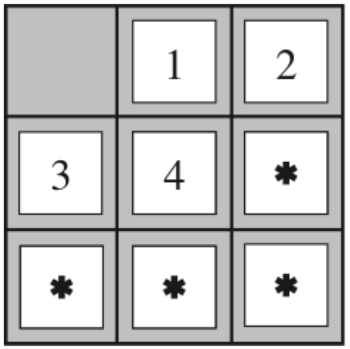
Estado inicial



Estado objetivo



Estado inicial

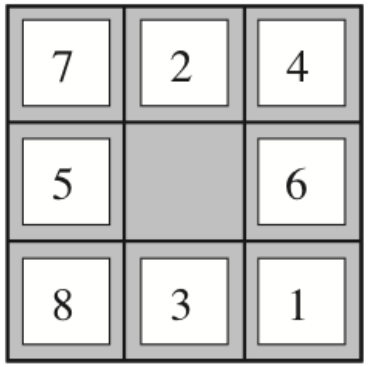


Estado objetivo

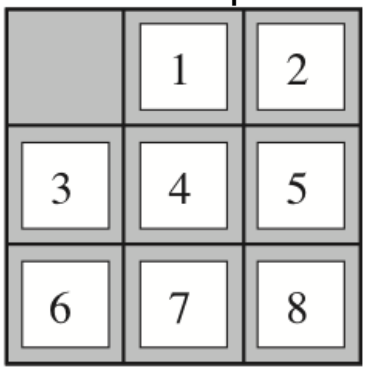
Universidad Zaragoza

Generar heurísticas: Bases de datos de patrones

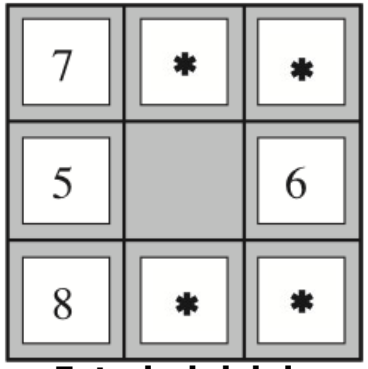
Podemos construir varias bases de patrones



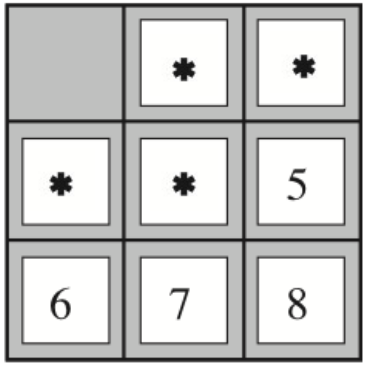
Estado inicial



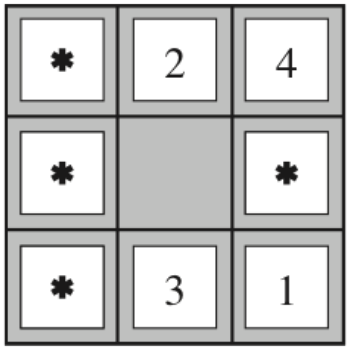
Estado objetivo



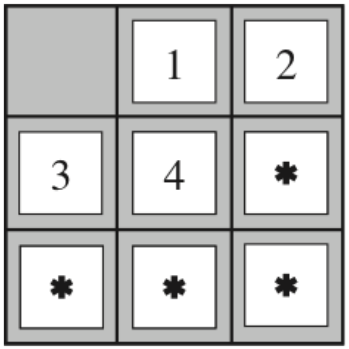
Estado inicial



Estado objetivo



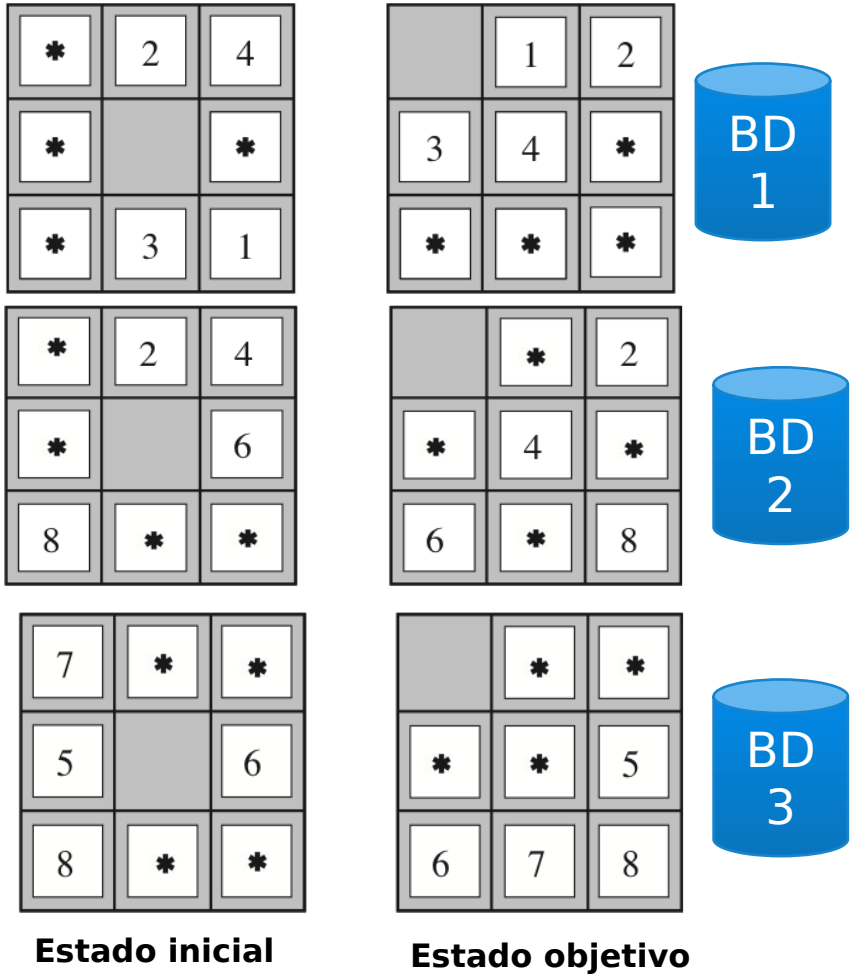
Estado inicial



Estado objetivo

Universidad Zaragoza

Generar heurísticas: Bases de datos de patrones



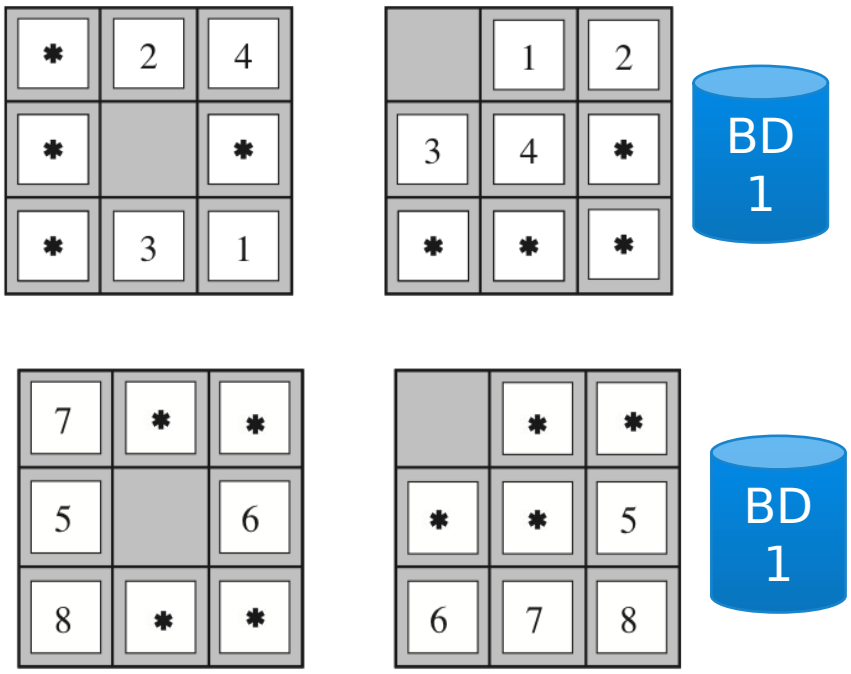
Combinamos con
$h(n) = \max\{h_1(n), \ldots h_m(n)\}$

Mejor que la distancia de Manhatan.

El número de nodos generados para el 15-puzle puede reducirse en un factor de 1000 comparado con la distancia de Manhatan.

Universidad Zaragoza

Generar heurísticas: Bases de datos de patrones disjuntas



Bases de datos disjuntas No podemos combinar h sumando las $\mathbf{h_i(n)}$, porque unos movimientos interfieren en los otros.

Pero si $h_1(n)$, no es el coste total del subproblema 1-2-3-4 sino el *número de movimientos que involucran 1-2-3-4*, y el de $h_2(n)$ es el *número de movimientos de 5-6-7-8* la suma es una cota inferior del coste total.

En el 15-puzzle se reducen los nodos generados en un orden de 10000 comparado con la distancia de Manhatan.

Para el cubo de Rubik se han propuesto otras formas de heurística aditiva pero no mejoran la Mejor heurística no aditiva.

Universidad Zaragoza
1542

Inteligencia Artificial

(30223) Grado en Ingeniería Informática

lección 4. Búsqueda informada (Heurística)
Sección 3.5 y 3.6 (AIMA)