Inteligencia Artificial

Resolución de problemas (III.2)

Búsqueda informada

Búsqueda informada

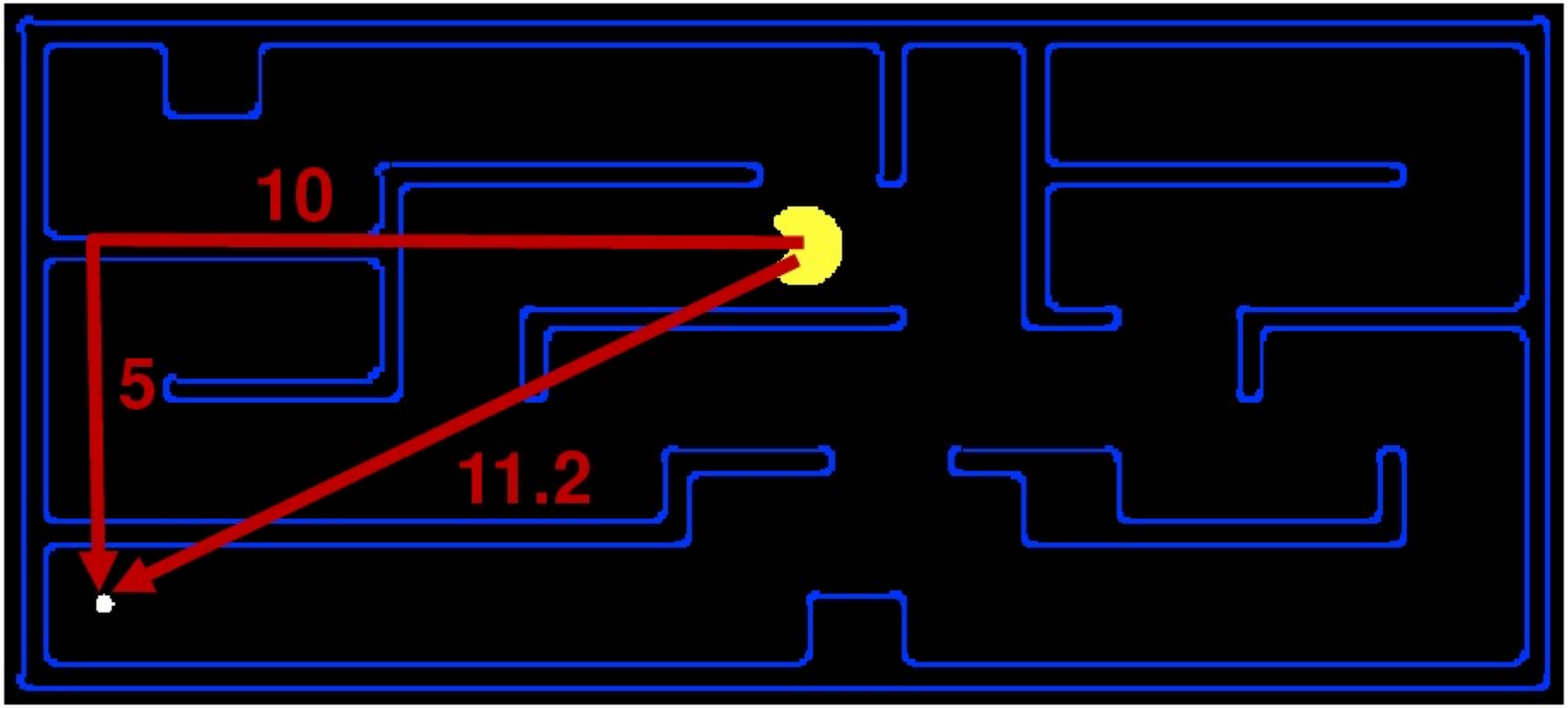
- Heurísticas
- Estrategias de búsqueda informada
  - Búsqueda voraz (greedy) o primero mejor
  - Búsqueda A*

Heurísticas

- La búsqueda informada mejora a la búsqueda no informada mediante la introducción en el proceso de búsqueda, de **información específica del problema** que permita acelerar la búsqueda.
- Una **heurística** *h(n)* es una estimación de lo cercano que se encuentra un nodo al objetivo
- Las heurísticas únicamente son válidas para un problema (no son generales).

Heurísticas en Pacman

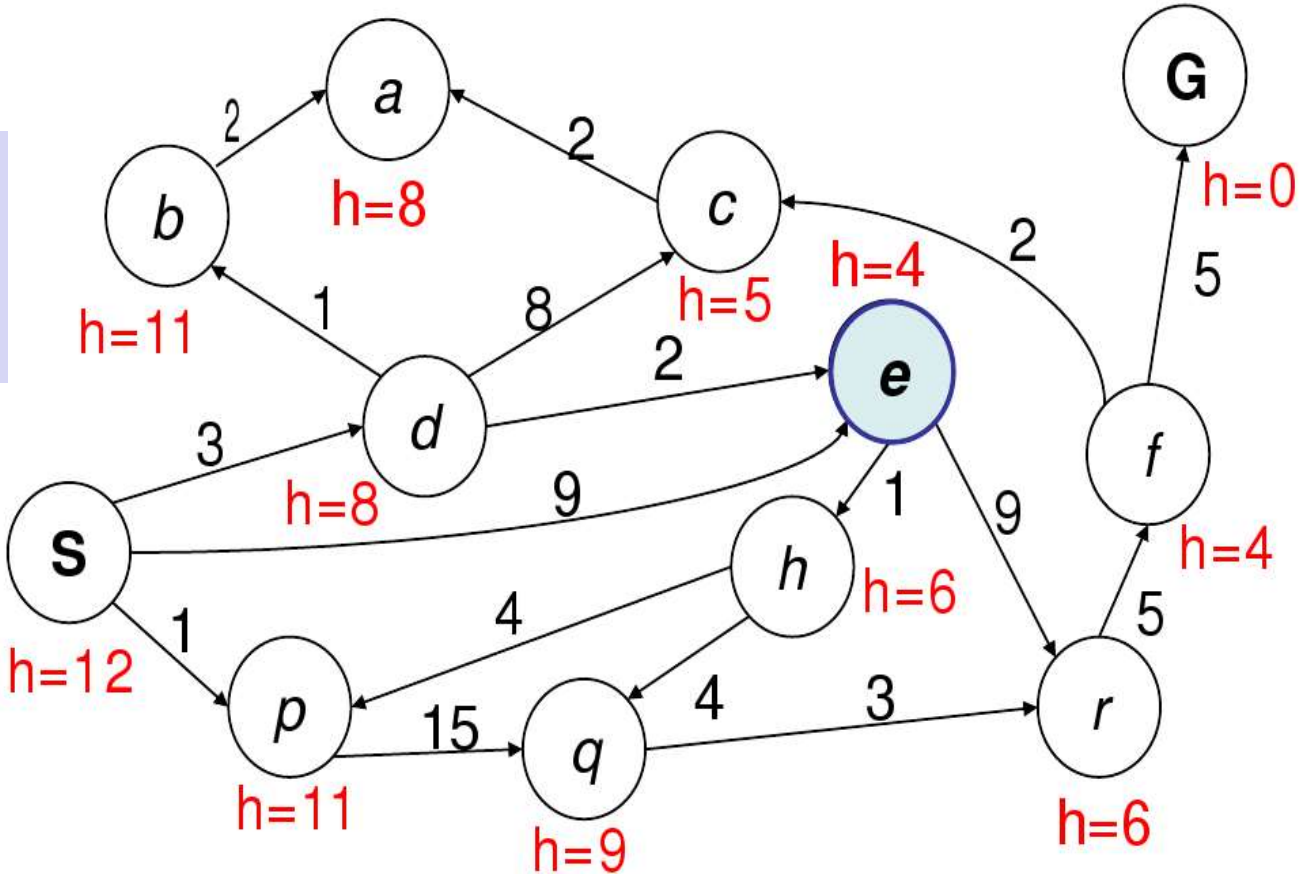
- Ejemplos: distancia Euclídea, distancia Manhattan.



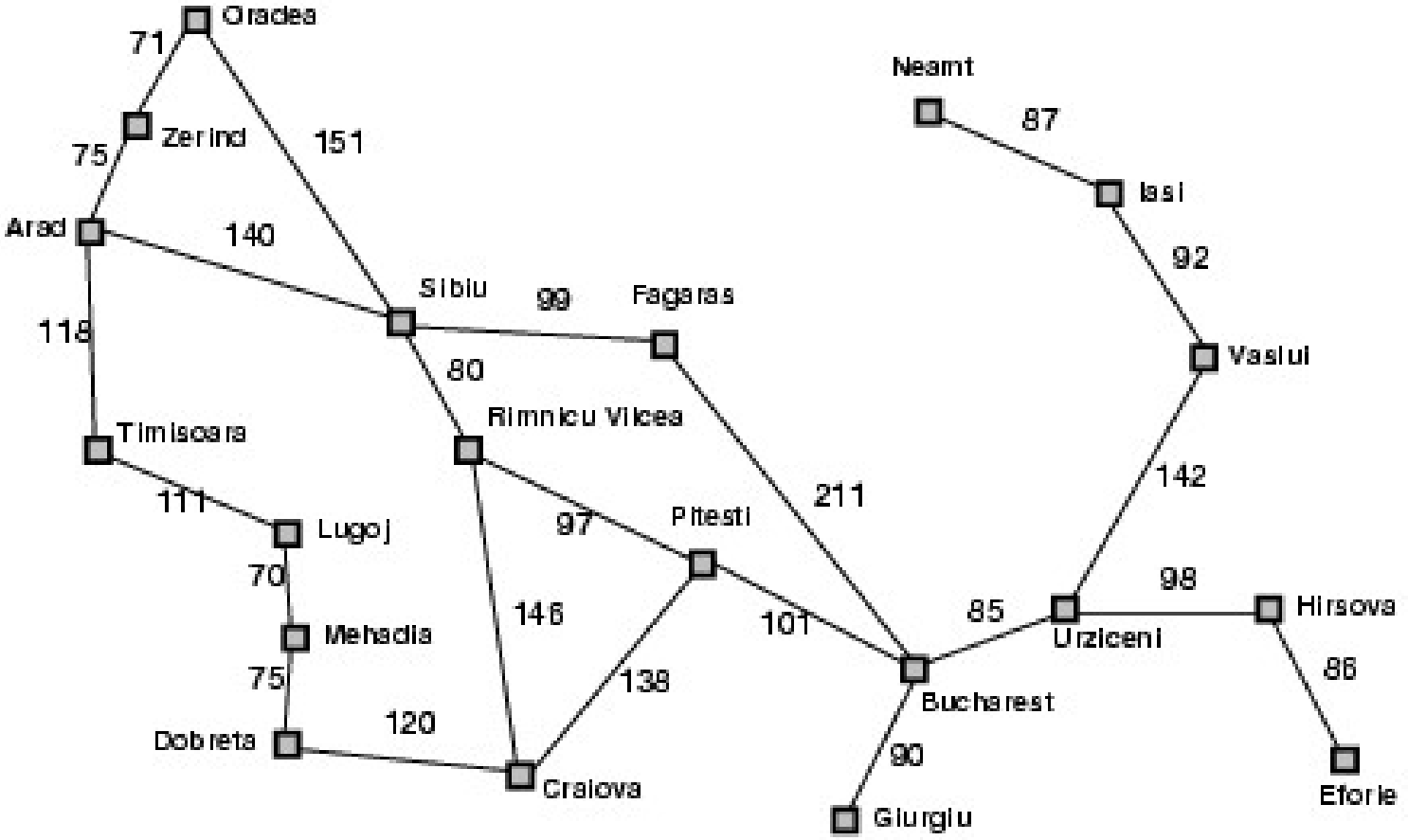
Búsqueda greedy primero mejor

- Expande el nodo que parece estar más cerca del objetivo (aquel que minimiza h(n))
- Función de evaluación f(n)=h(n)

Solución?
- Nodo inicial: **e**
- Nodo objetivo: **G**



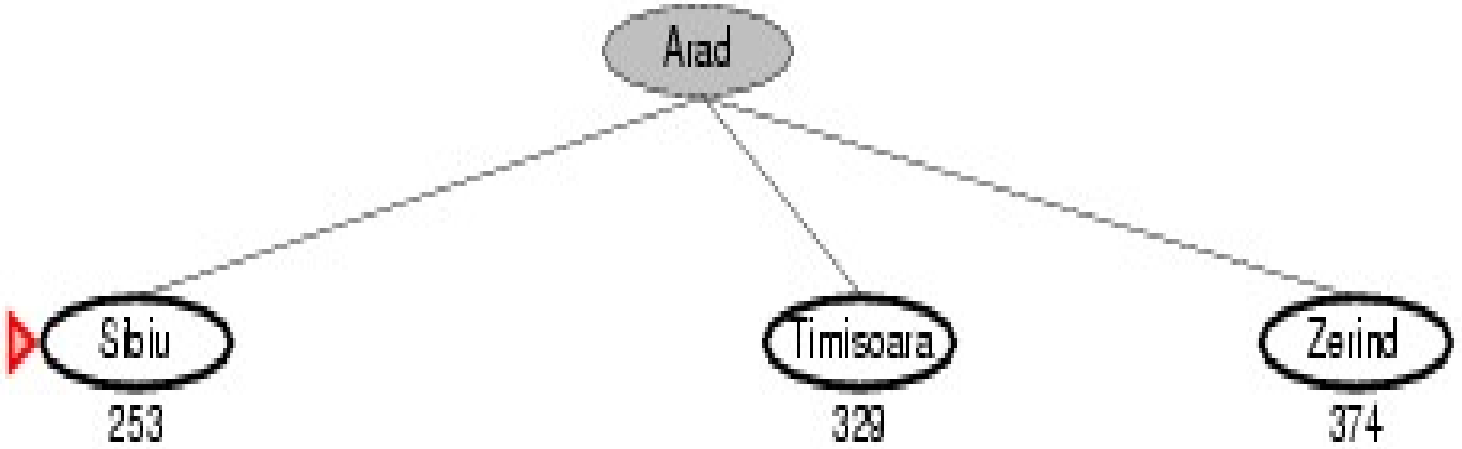
Romania (distancias en km)



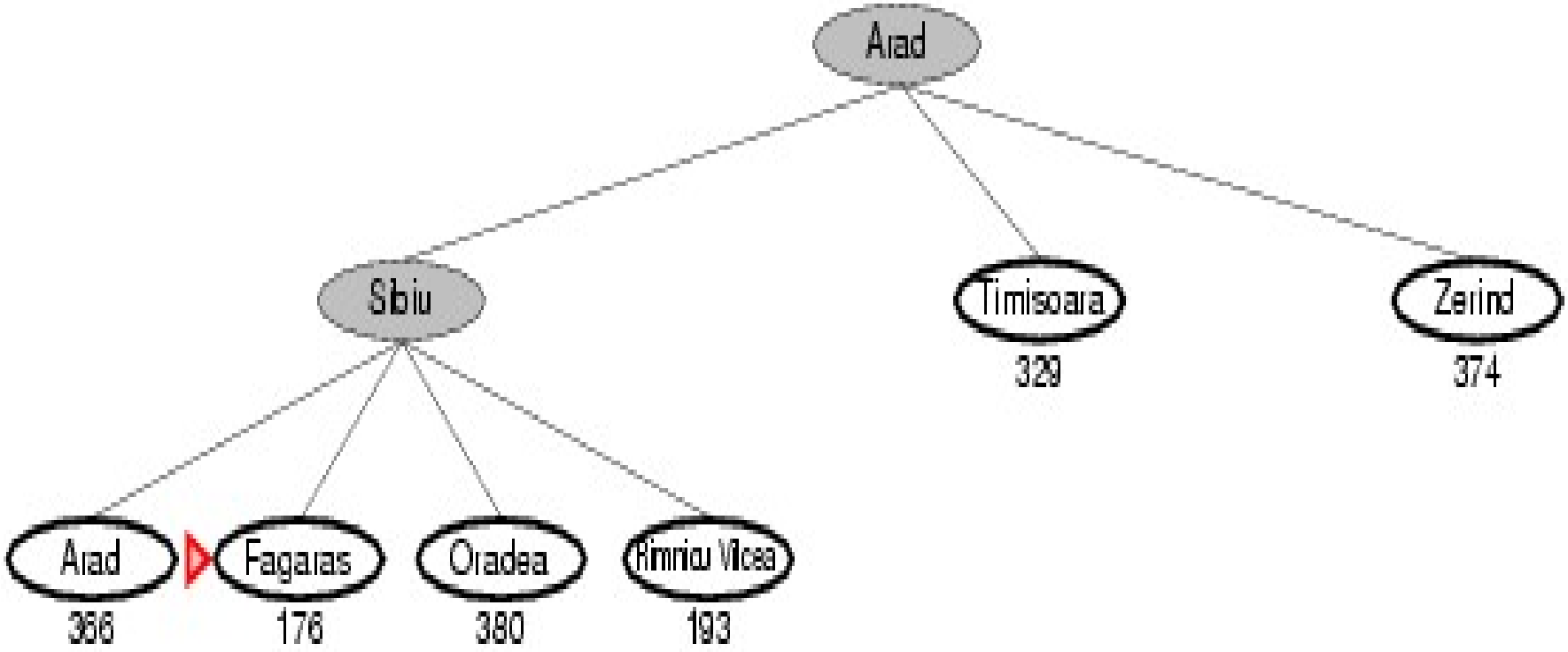
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Ej. búsqueda greedy primero mejor

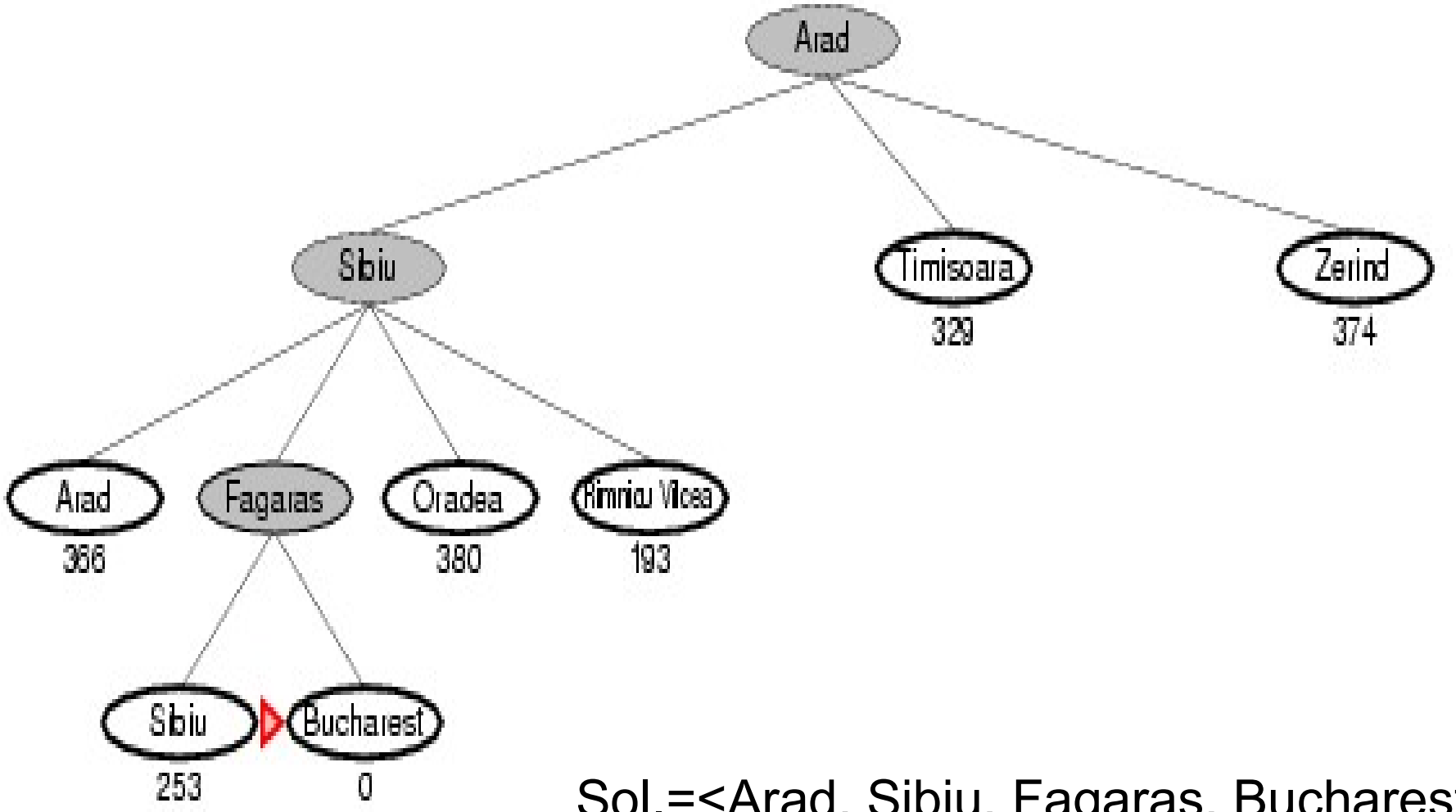
Ej. búsqueda greedy primero mejor



Ej. búsqueda greedy primero mejor



Ej. búsqueda greedy primero mejor



Sol.=<Arad, Sibiu, Fagaras, Bucharest>
Coste= 140 + 99 + 211 = 450

Propiedades de la búsqueda greedy primero mejor

UNIVERSITAT DE BARCELONA

- Completa? No – puede quedarse atrapada en blucles,
  - p.ej., al ir de Iasi a Fagaras: Iasi → Neamt → Iasi → Neamt ...
- Tiempo? $O(b^m)$, pero una buena heurística puede proporcionar una mejora dramática.
- Espacio? $O(b^m)$ – mantiene todos los nodos en memoria
- Óptima? No

Propiedades de la búsqueda greedy primero mejor

UNIVERSITAT DE BARCELONA

- Completa? No – puede quedarse atrapada en blucles,
  - p.ej., al ir de Iasi a Fagaras: Iasi → Neamt → Iasi → Neamt ...
- Tiempo? $O(b^m)$, pero una buena heurística puede proporcionar una mejora dramática.
- Espacio? $O(b^m)$ – mantiene todos los nodos en memoria
- Óptima? No

  Sol.=<Arad, Sibiu, Rimmicu Vikea, Pitesti, Bucharest>

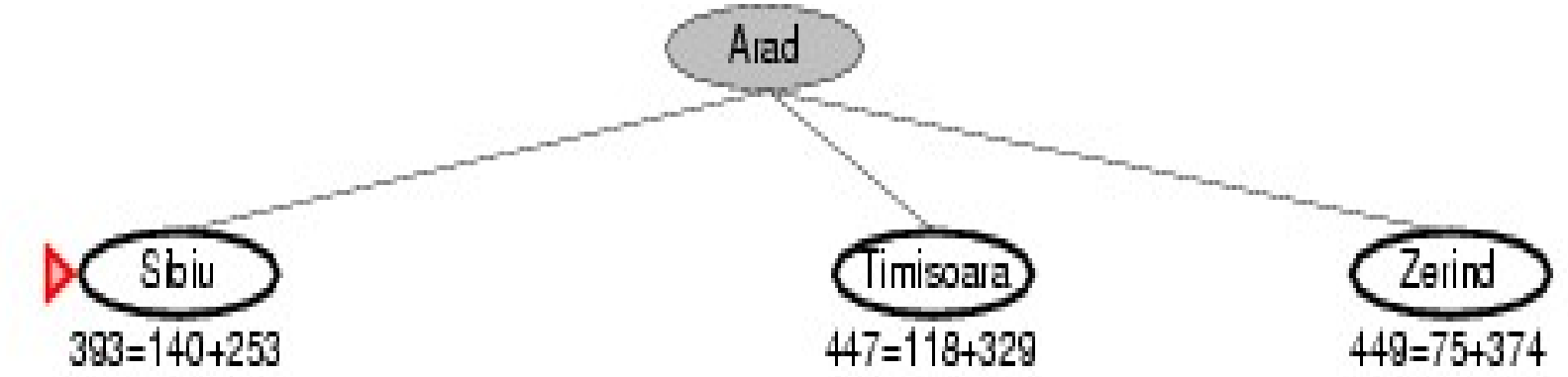
  Coste= 140 + 80 + 97 + 101 = 418

Búsqueda A*

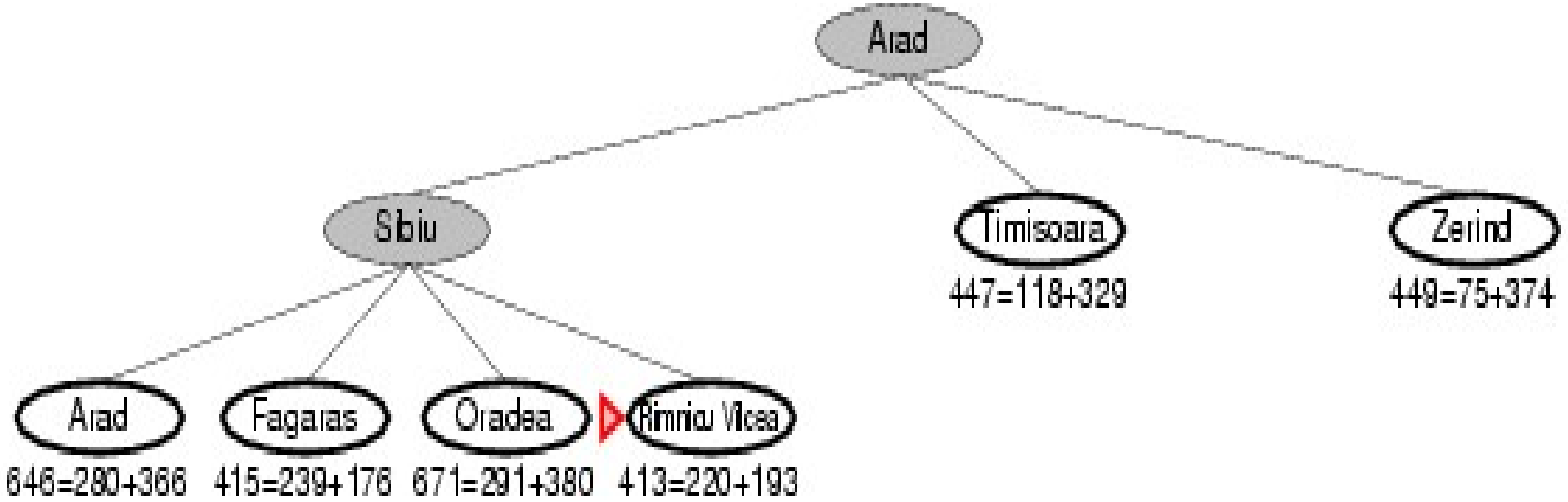
- Idea: evitar expandir caminos que ya son caros
- Función de evaluación $f(n) = g(n) + h(n)$
- $g(n)$ = coste de alcanzar n desde el estado inicial
- $h(n)$ = coste estimado desde n hasta el objetivo
- $f(n)$ = coste estimado de la mejor solución que pase por n

Ejemplo de búsqueda A*

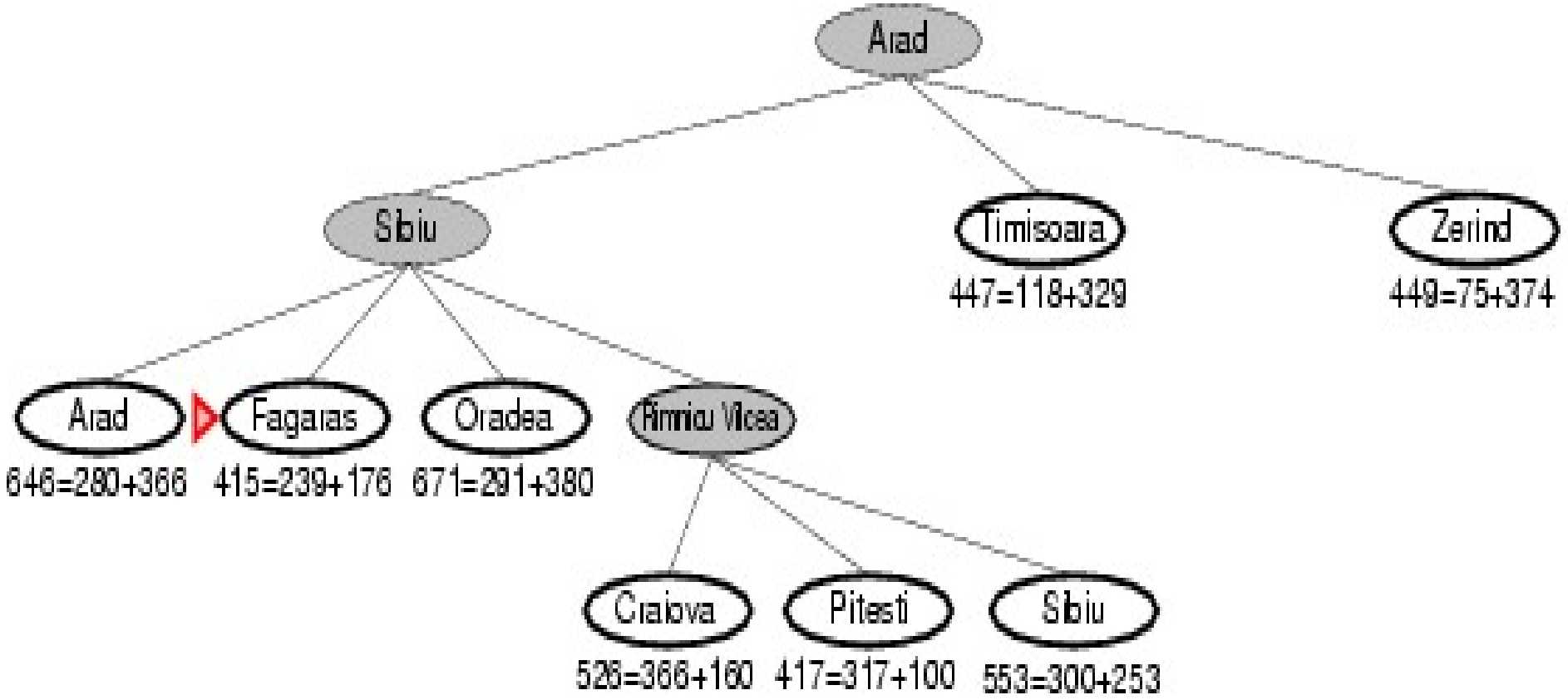
Ejemplo de búsqueda A*



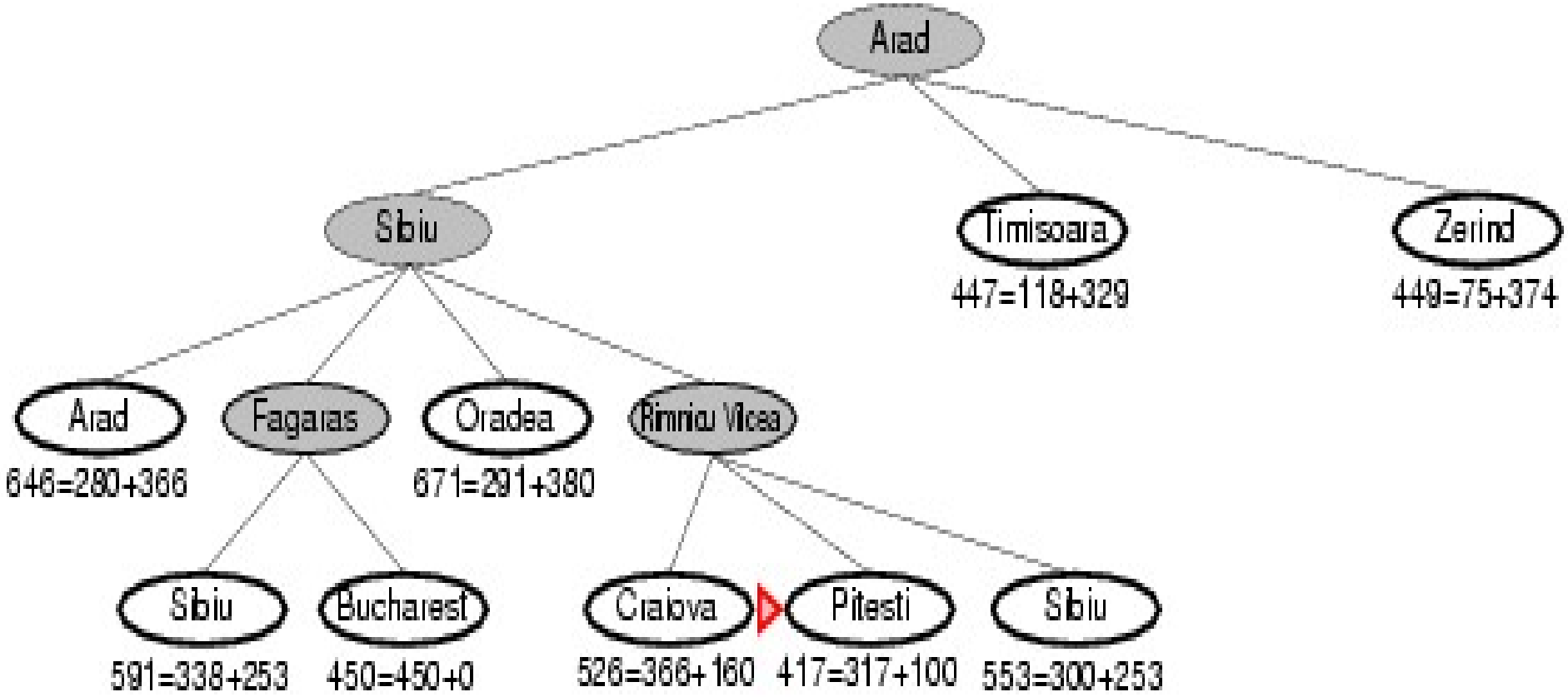
Ejemplo de búsqueda A*



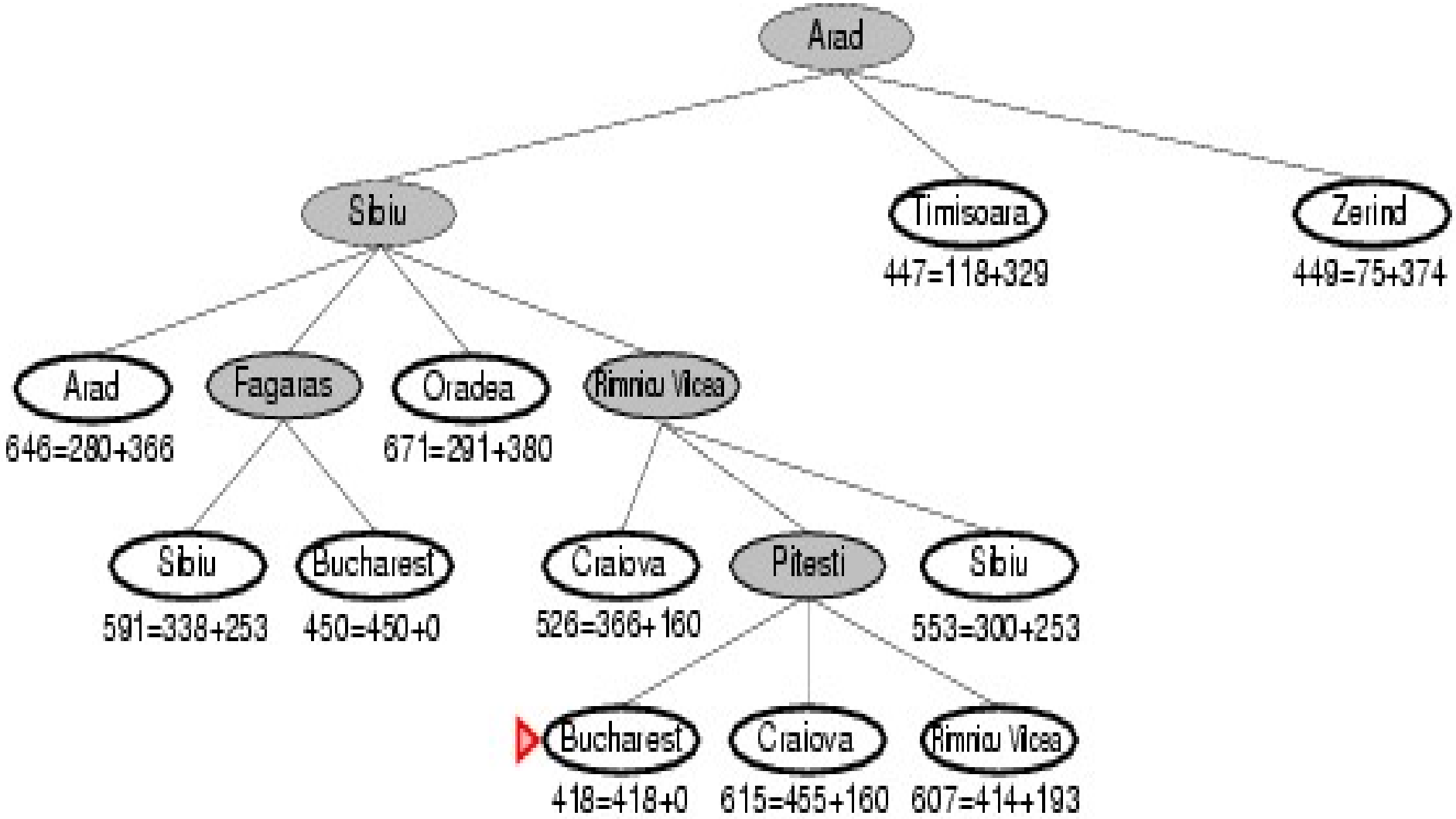
Ejemplo de búsqueda A*



Ejemplo de búsqueda A*



Ejemplo de búsqueda A*



Heurísticas admisibles

- Una heurística *h(n)* es admisible si para todo nodo *n*, $h(n) \leq h^*(n)$, donde $h^*(n)$ es el coste real de alcanzar el estado objetivo desde *n*.

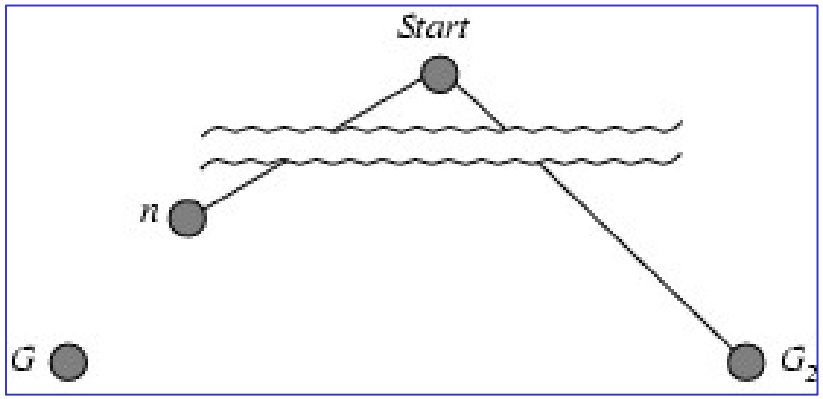
- Una heurística admisible nunca sobre-estima el coste de alcanzar el objetivo, es decir, es optimista

- Ejemplo: $h_{SLD}(n)$ nunca sobre-estima la distancia por carretera

Teorema: Si *h(n)* es admisible, la búsqueda A* usando ***búsqueda en árbol*** es óptima

Optimalidad de A* (prueba)

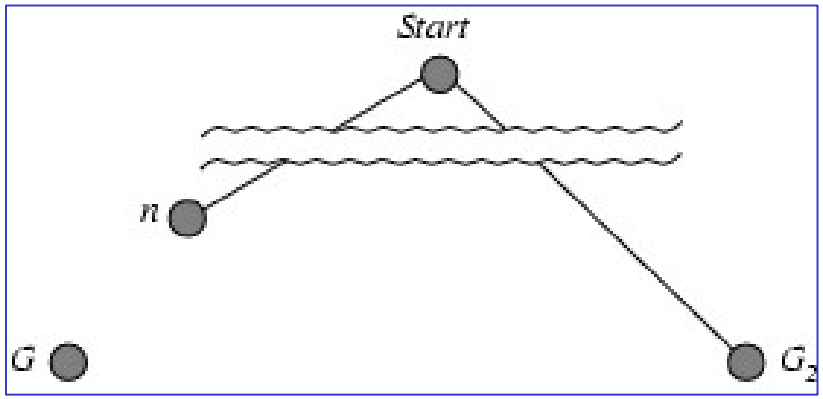
- Suponer que en la frontera se ha generado un objetivo sub-óptimo G_2 .
- Sea *n* un nodo no expandido en la frontera que pertenece a un camino óptimo.



- $f(G_2) = g(G_2)$ ya que $h(G_2) = 0$
- $g(G_2) > g(G)$ ya que G_2 es subóptimo
- $f(G) = g(G)$ ya que $h(G) = 0$
- Por tanto: $f(G_2) > f(G)$

Optimalidad de A* (prueba)

- Suponer que en la frontera se ha generado un objetivo sub-óptimo G_2 .
- Sea *n* un nodo no expandido en la frontera que pertenece a un camino óptimo.



- $f(G_2) = g(G_2)$ ya que $h(G_2) = 0$
- $g(G_2) > g(G)$ ya que G_2 es subóptimo
- $f(G) = g(G)$ ya que $h(G) = 0$
- Por tanto: $f(G_2) > f(G)$
- $h(n) \leq h^*(n)$ ya que h es admisible
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) \leq f(G)$

Por lo tanto,

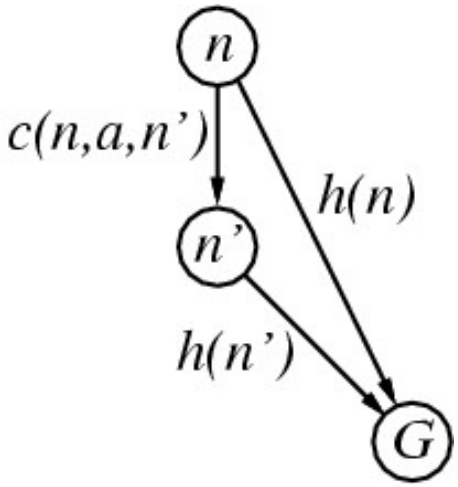
$$f(G_2) > f(n)$$

y A* nunca seleccionará G_2 para expansión (seleccionará n, el óptimo)

Heurísticas consistentes

- Una heurística es consistente si para cada nodo *n*, todo sucesor *n'* de *n* generado por cualquier acción *a*, cumple que:

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



- Si *h* es consistente, tenemos que

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

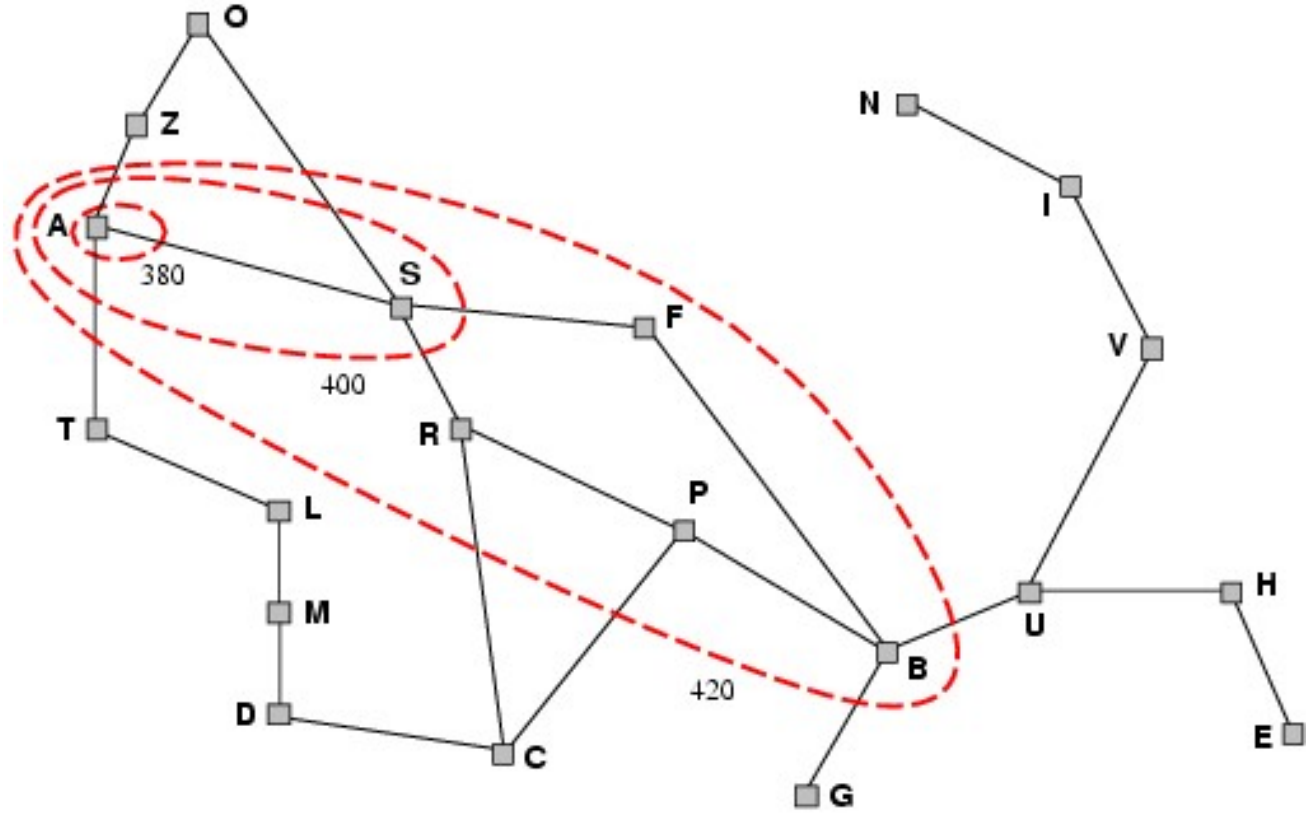
- Es decir, *f(n)* es no decreciente a lo largo de cualquier camino

Teorema: Si *h(n)* es **consistente**, la búsqueda **A*** utilizando ***búsqueda en grafos*** es **óptima**

Optimalidad de A*

- A* expande nodos en orden de incremento del valor de *f*

- Gradualmente añade "*f*-contornos" de nodos
- Contorno *i* tiene todos los nodos con $f=f_i$, donde $f_i < f_{i+1}$



Propiedades de A*

- Completo? Sí (a nos ser que haya infinitos nodos con f ≤ *f(G)*)
- Tiempo? Exponencial
- Espacio? Mantiene todos los nodos en memoria
- Óptimo? Sí

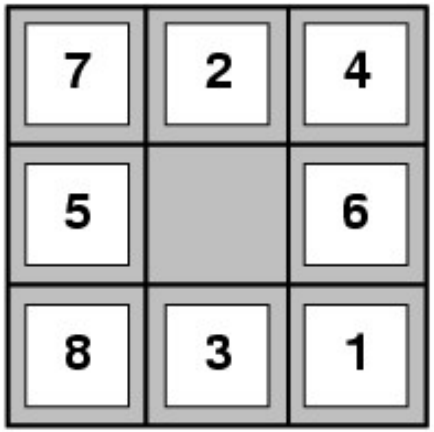
Variantes A*

UNIVERSITAT DE BARCELONA

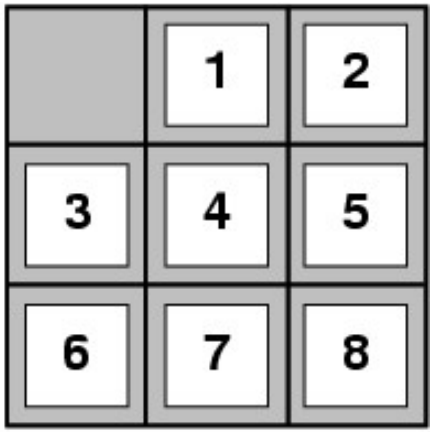
- Algoritmos con limitación de uso de memoria:
  - Iterative Deapening A* (IDA*)
  - Recursive Best First Search (RBFS)
  - Memory Bounded A* (MA*)
  - Simplified MA* (SMA*)
  - ….
  - IJCAI'11: http://ijcai-11.iiia.csic.es/
    http://ijcai.org/papers11/contents.php
    - Ex.: Planning with SAT, Admissible Heuristics and A* (Australia)

Heurísticas admisibles

Para el 8-puzzle?



Start State

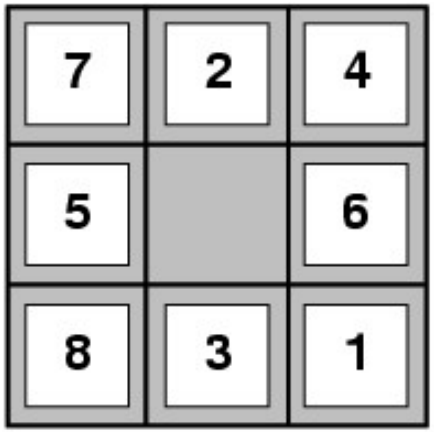


Goal State

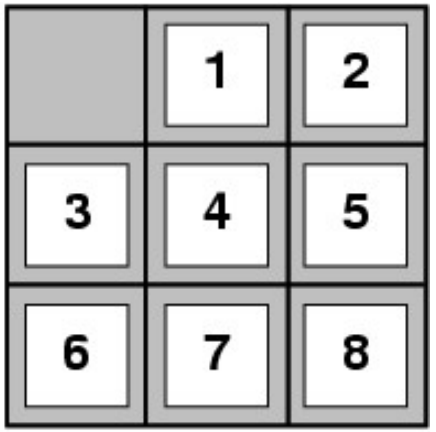
Heurísticas admisibles

Para el 8-puzzle:

- $h_1(n)$ = número de cuadrados fuera de sitio
- $h_2(n)$ = distancia total de Manhattan (es decir, número de movimientos desde la posición actual hasta la posición objetivo para cada cuadrado)



Start State



Goal State

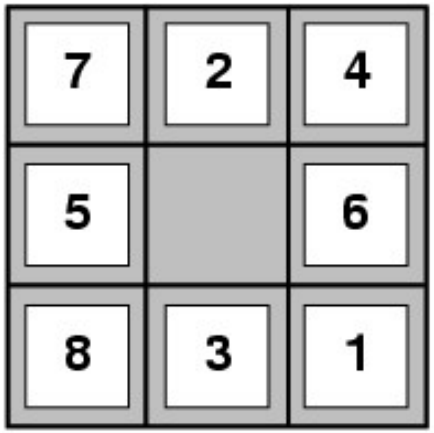
$h_1(S) = ?$

$h_2(S) = ?$

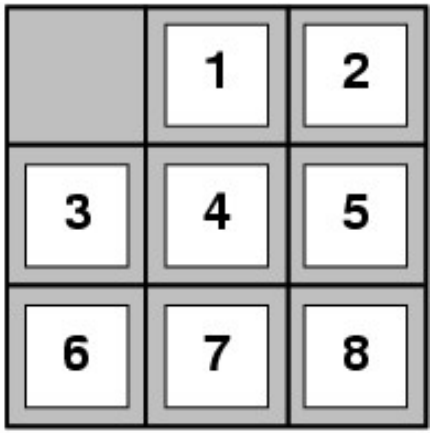
Heurísticas admisibles

Para el 8-puzzle:

- $h_1(n)$ = número de cuadrados fuera de sitio
- $h_2(n)$ = distancia total de Manhattan (es decir, número de movimientos desde la posición actual hasta la posición objetivo para cada cuadrado)



Start State



Goal State

$h_1(S) = ?$ 8

$h_2(S) = ?$ 3+1+2+2+3+2+2+3= 18

7 2 4 5 6 8 3 1

Dominancia

- Si $h_2(n) \geq h_1(n)$ para todo n (siendo ambas admisibles) entonces h_2 domina h_1

- h_2 es mejor para buscar

- Costes típicos de búsqueda (número medio de nodos expandidos):

- *d=12* IDS = 3,644,035 nodos
  $A^*(h_1)$ = 227 nodos
  $A^*(h_2)$ = 73 nodos

- *d=24* IDS = demasiados nodos
  $A^*(h_1)$ = 39,135 nodos
  $A^*(h_2)$ = 1,641 nodos

Problemas relajados

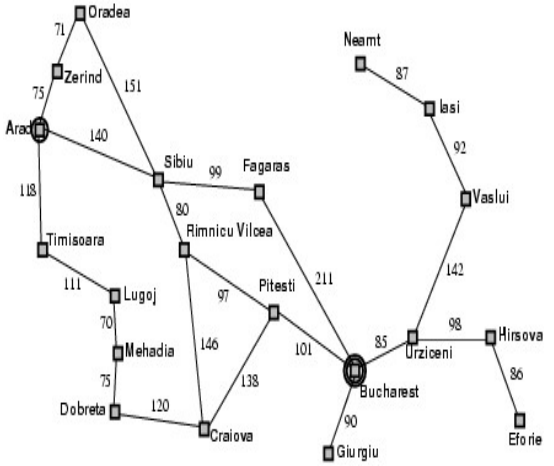
- Un problema con menos restricciones en las acciones se denomina problema relajado.
- El coste de una solución óptima a un problema relajado es una heurística admisible del problema original.
- Si las reglas del 8-puzzle se relajan de forma que un cuadrado se pueda mover a cualquier parte, entonces $h_1(n)$ proporciona la solución más corta.
- Si las reglas se relajan de forma que un cuadrado se puede mover a cualquier cuadrado adyacente, entonces $h_2(n)$ nos da la solución más corta.

Resumen (hasta ahora)

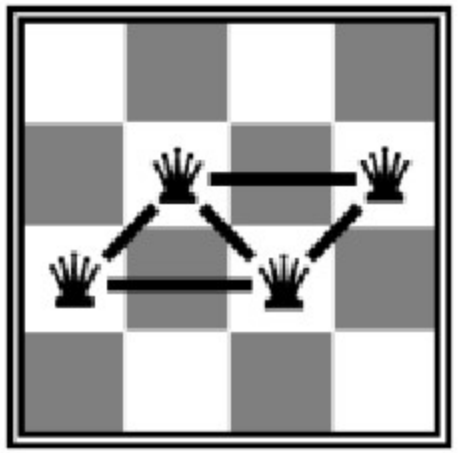
- Las heurísticas nos permiten introducir información particular del problema para guiar la búsqueda
- Si únicamente utilizamos la información de cuánto nos queda para llegar → primero mejor greedy
- Si unimos a lo que nos queda para llegar cuánto llevamos recorrido → A*
- A* es óptimo con heurísticas admisibles (y en grafos con heurísticas consistentes)
- El diseño de buenas heurísticas es clave. Se pueden utilizar problemas relajados.

Tipos de problemas

- Problemas de planificación
  - Queremos encontrar un camino a la solución (habitualmente el camino óptimo)



- Problemas de identificación
  - Sólo queremos encontrar un estado que cumpla con el objetivo (habitualmente el estado óptimo)



Algoritmos de búsqueda local

- Búsqueda local: Partimos de una situación y tratamos de mejorarla hasta convertirla en óptima
- Generalmente son mucho más eficientes que los algoritmos de búsqueda completos... pero no son completos.
- Los algoritmos de búsqueda local permiten encontrar soluciones subóptimas en problemas muy complejos.

Ascenso de colinas

Empieza donde quieras

Mientras alguno de los vecinos del estado actual es mejor que él

El vecino pasa a ser el estado actual

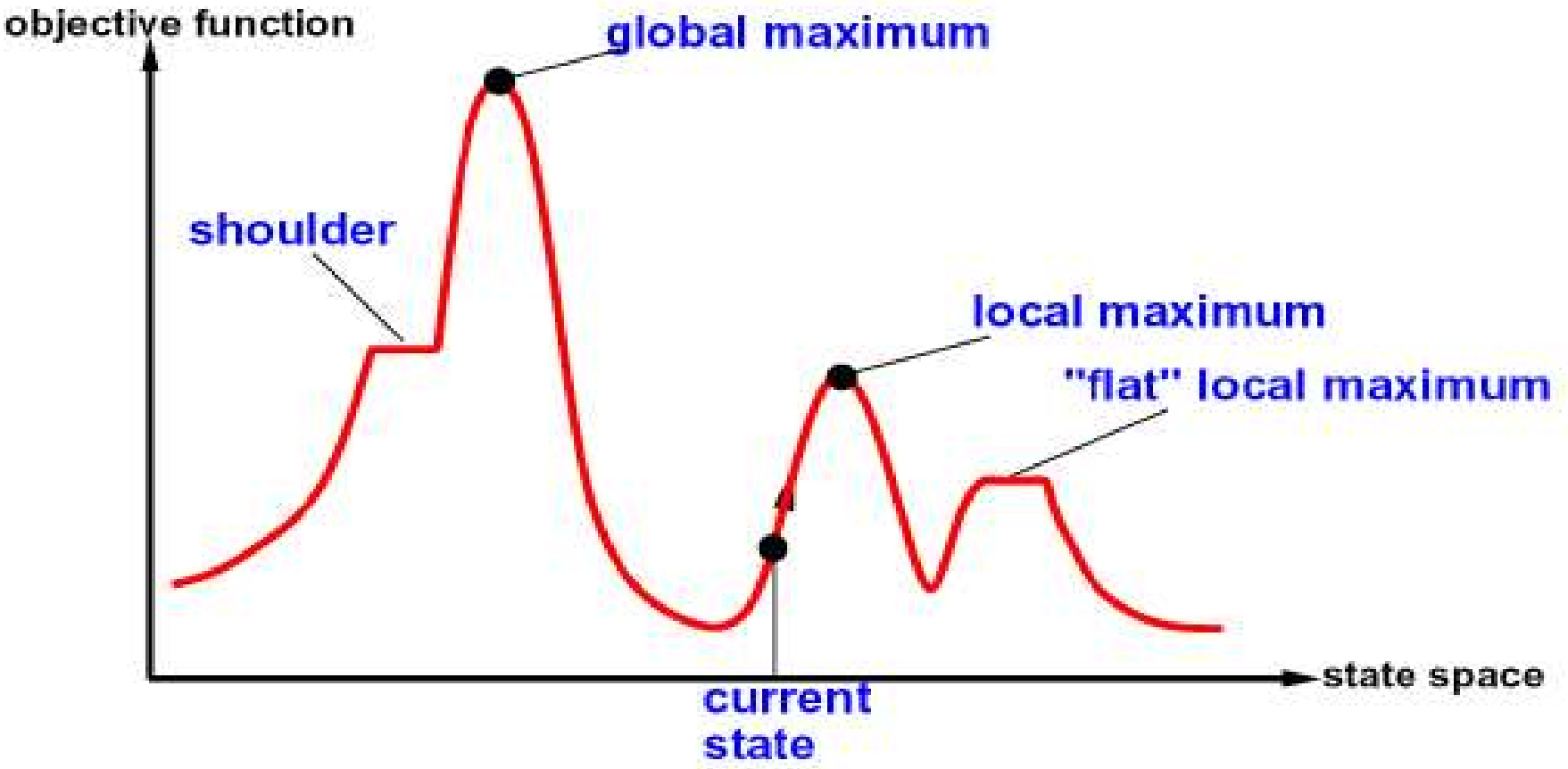
- ¿Por qué puede ser muy mala idea?
  - ¿Es completo?
  - ¿Es óptimo?
- ¿Qué tiene de bueno?

Ascenso de colinas

```
función ASCENSION-COLINAS(problema) devuelve un estado que es un máximo local
  entradas: problema, un problema
  variables locales: actual, un nodo
                     vecino, un nodo

  actual ← HACER-NODO(ESTADO-INICIAL[problema])
  bucle hacer
    vecino ← sucesor de valor más alto de actual
    si VALOR[vecino] ≤ VALOR[actual] entonces devolver ESTADO[actual]
    actual ← vecino
```

Ascenso de colinas: Diagrama



- ¿Reinicios aleatorios?
- ¿Pasos laterales aleatorios?

Temple simulado

función TEMPLE-SIMULADO(*problema, esquema*) **devuelve** un estado solución
  entradas: *problema*, un problema
    esquema, una aplicación desde el tiempo a «temperatura»
  variables locales: *actual*, un nodo
    siguiente, un nodo
    T, una «temperatura» controla la probabilidad de un paso hacia abajo

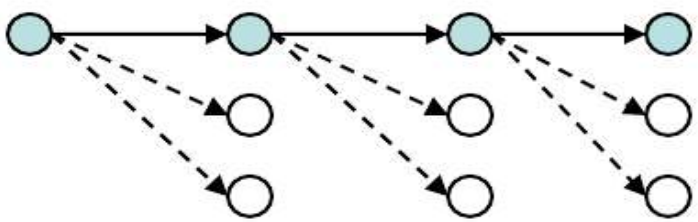
  actual ← HACER-NODO(ESTADO-INICIAL[*problema*])
  para $t \leftarrow 1$ **a** ∞ **hacer**
    $T \leftarrow$ *esquema*[t]
    si $T = 0$ **entonces devolver** *actual*
    siguiente ← un sucesor seleccionado aleatoriamente de *actual*
    $\Delta E \leftarrow$ VALOR[*siguiente*] − VALOR[*actual*]
    si $\Delta E > 0$ **entonces** *actual* ← *siguiente*
    en caso contrario *actual* ← siguiente sólo con probabilidad $e^{\Delta E/T}$

Temple simulado

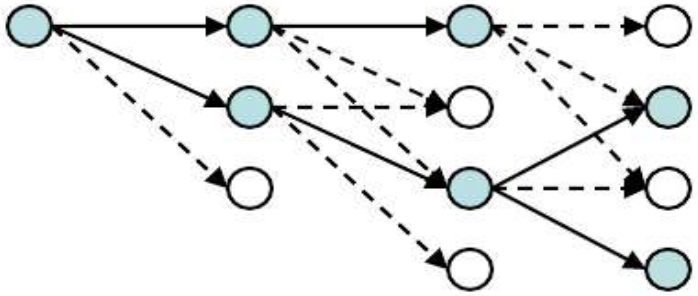
- Garantía teórica:
  - Si T se reduce poco a poco, convergerá a un estado óptimo.
- Realmente, si son necesarios muchos pasos negativos para escapar de un máximo local, se hace improbable que ocurran

Búsqueda por haz local

- Es como la búsqueda voraz pero mantiene *k* estados en cada momento



Greedy Search



Beam Search

- Variables: Tamaño del haz, ¿diversidad?
- Es la mejor elección en muchos problemas
- ¿Completa? ¿Óptima?
- ¿Por qué criterio deben ordenarse los nodos?

Algoritmos Genéticos

- k estados: población de k individuos
- Los sucesores se generan combinando dos estados padre (reproducción sexual)
- Se seleccionan los mejores

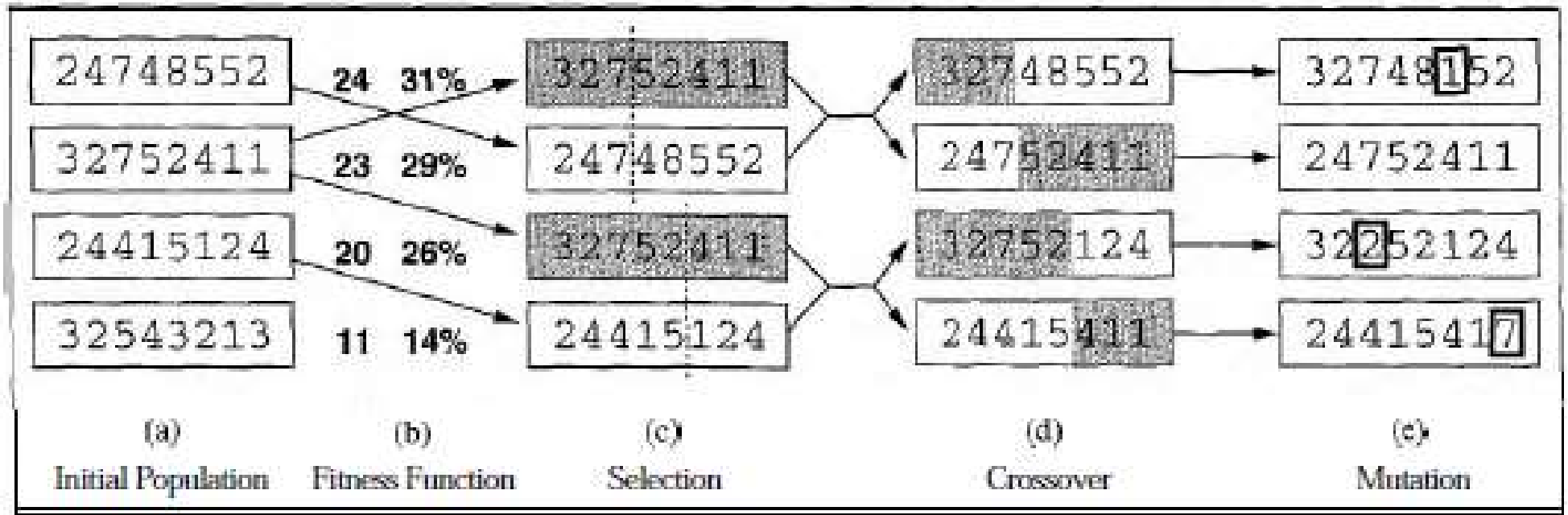


Figure 4.15 The genetic algorithm. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Resumen

- Las heurísticas nos permiten introducir información particular del problema para guiar la búsqueda
- Si únicamente utilizamos la información de cuánto nos queda para llegar → primero mejor greedy
- Si unimos a lo que nos queda para llegar cuánto llevamos recorrido → A*
- A* es óptimo con heurísticas admisibles (y en grafos con heurísticas consistentes)
- El diseño de buenas heurísticas es clave. Se pueden utilizar problemas relajados.
- Los algoritmos de búsqueda local permiten encontrar soluciones subóptimas a problemas muy complejos.