Tema 5

Búsqueda Heurística

elisa.guerrero@uca.es

Tema 5. Búsqueda Heurística
Objetivos

- Al finalizar este tema el alumno deberá ser capaz de:
  - Definir funciones heurísticas apropiadas a los problemas planteados
  - Aplicar los algoritmos de búsqueda heurística
  - Aplicar los algoritmos de búsqueda local
  - Evaluar las ventajas de cada método
  - Seleccionar la mejor estrategia de acuerdo a las características del problema
  - Implementar todas las estrategias en un lenguaje de programación

Tema 5. Búsqueda Heurística
Índice del Tema

1. Introducción
Heurística

HEURÍSTICA

- Manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc (RAE, http://dle.rae.es)
- Proceso que puede resolver un problema dado, pero que no ofrece ninguna garantía de que lo hará (Newell, Shaw y Simon, 1963).

FUNCIÓN HEURÍSTICA h(n)

- Dado un nodo n estima el coste de llegar desde n al nodo objetivo, siendo h(objetivo) = 0.
  - Utiliza información del dominio específico del problema.
  - No garantiza el éxito, pero suele ser mejor que la búsqueda a ciegas.

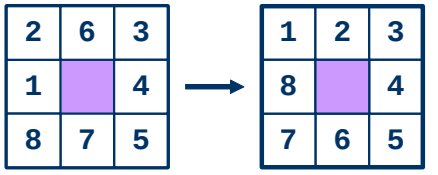
1. Introducción
Ejemplo: Heurísticas para el 8-puzle

- h1 = nº de piezas mal colocadas

  h1(x) = fichas(1,2,6,7,8)=5

- h2 = suma de las distancias de Manhattan de las posiciones a sus objetivos

  La distancia de Manhattan es el nº de filas y columnas que restan de la posición actual de una pieza a su posición final. Por ejemplo, la distancia Manhattan de la pieza 2 sería de 1, de la pieza 5 sería 0, etc.



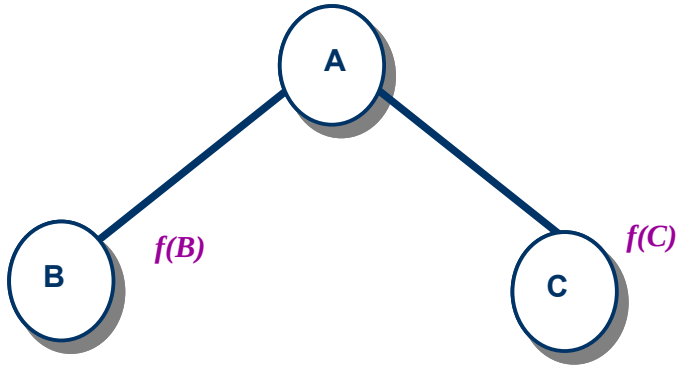
h2(x) = 1+1+0+0+0+2+1+1=6

2. Búsquedas Primero el Mejor (Best-First Search)

- La decisión final de expandir un nodo se basa en la función de evaluación $f(n)$:
  - $f(n)$ evalúa si el estado actual es el mejor estado para seguir expandiendo el árbol o grafo de búsqueda

Búsqueda Voraz o Avara
(Greedy search)

Algoritmo A*

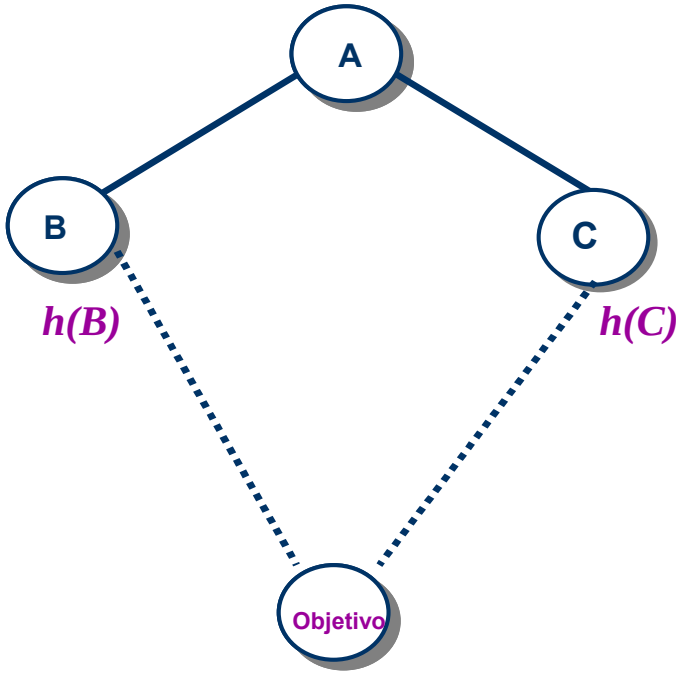


2. Búsquedas Primero el Mejor (Avara y A*)

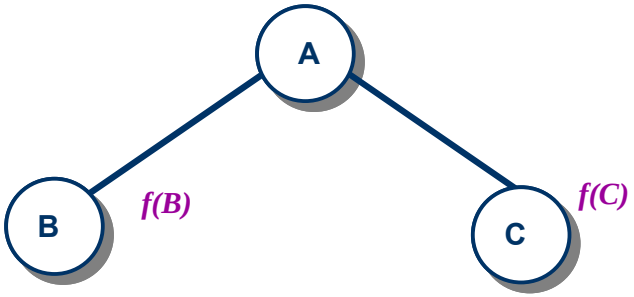
- Normalmente se establece el criterio de elegir el nodo con $f(n)$ de menor valor.
- Ordenan ascendentemente la lista ABIERTOS según el valor de $f(n)$ asociado a cada nodo.
- Combinan las ventajas de:
  - La Búsqueda en Profundidad:
    - Sigue un único camino, sin necesidad de generar todos los caminos posibles.
  - Y la Búsqueda en Anchura:
    - No se queda en bucles infinitos o caminos sin salida.

2. Búsquedas 1º el mejor heurística versus evaluación

Una **función heurística** estima el coste de alcanzar el objetivo desde un estado *n*



Una **función de evaluación** describe la conveniencia de expandir el nodo *n*



2.1 Búsqueda Voraz (Avara o Greedy Search)

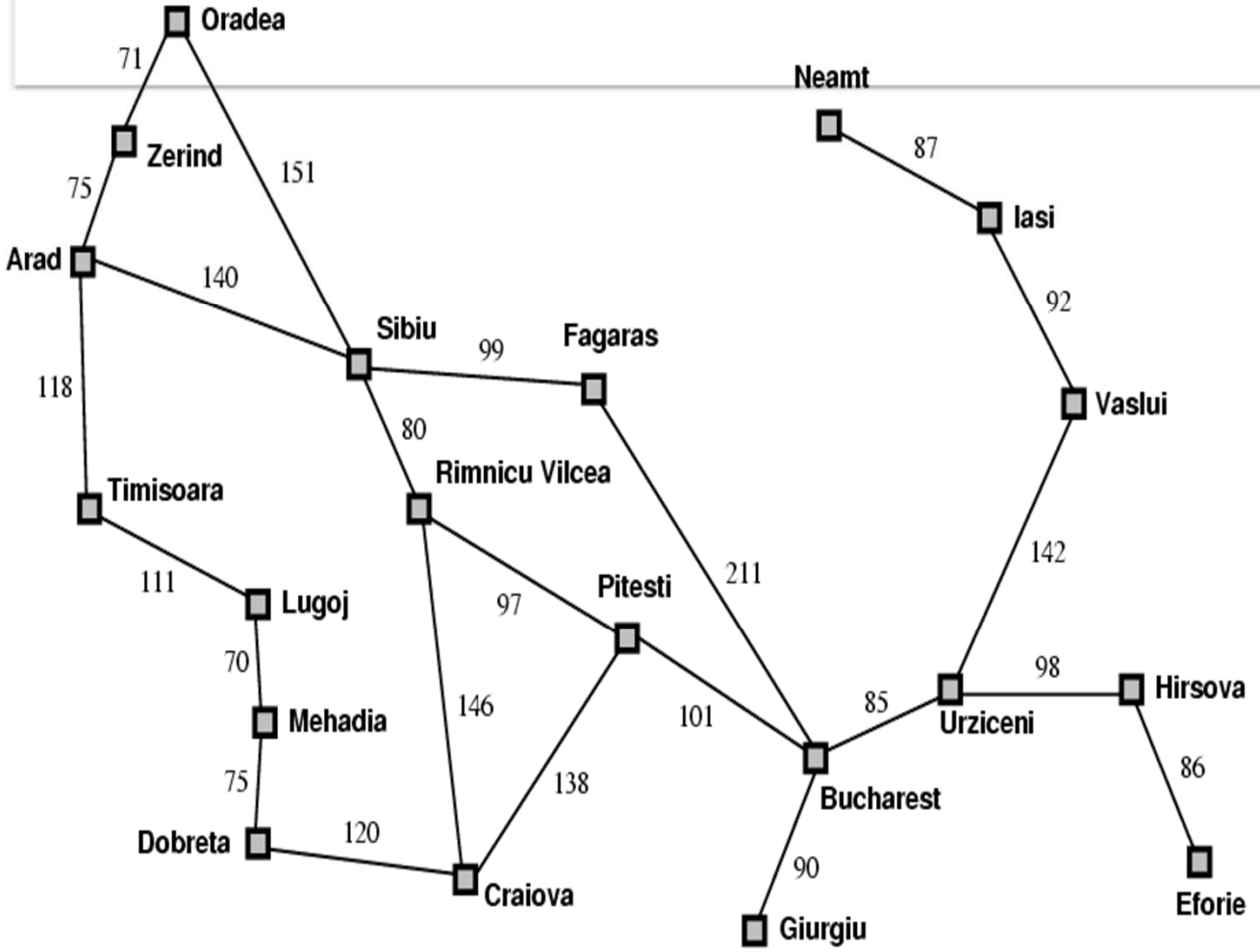
- Selecciona de la lista ABIERTOS el nodo con el menor valor de f(n), siendo:

$$f(n) = h(n)$$

- Se pretende llegar rápidamente a la solución sin importar tanto el coste.

- Trata de expandir el nodo más cercano al objetivo → minimizar el coste estimado para alcanzar el estado final, pero no tiene en cuenta el coste real de llegar hasta *n.*

Distancia entre ciudades rumanas

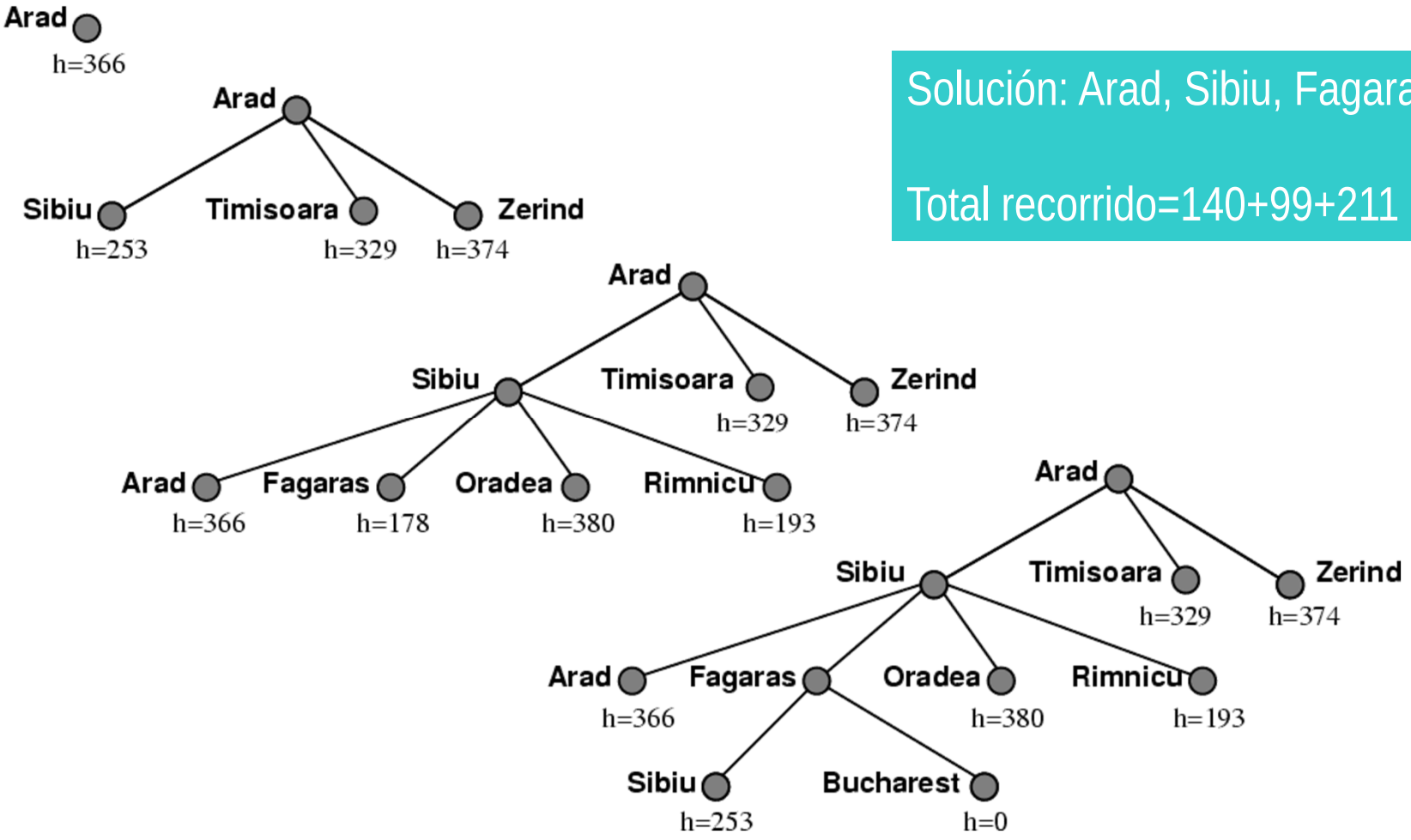


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Encontrar el camino más corto por carretera desde Arad a Bucarest

h(n): distancia en línea recta de la ciudad *n* a Bucarest

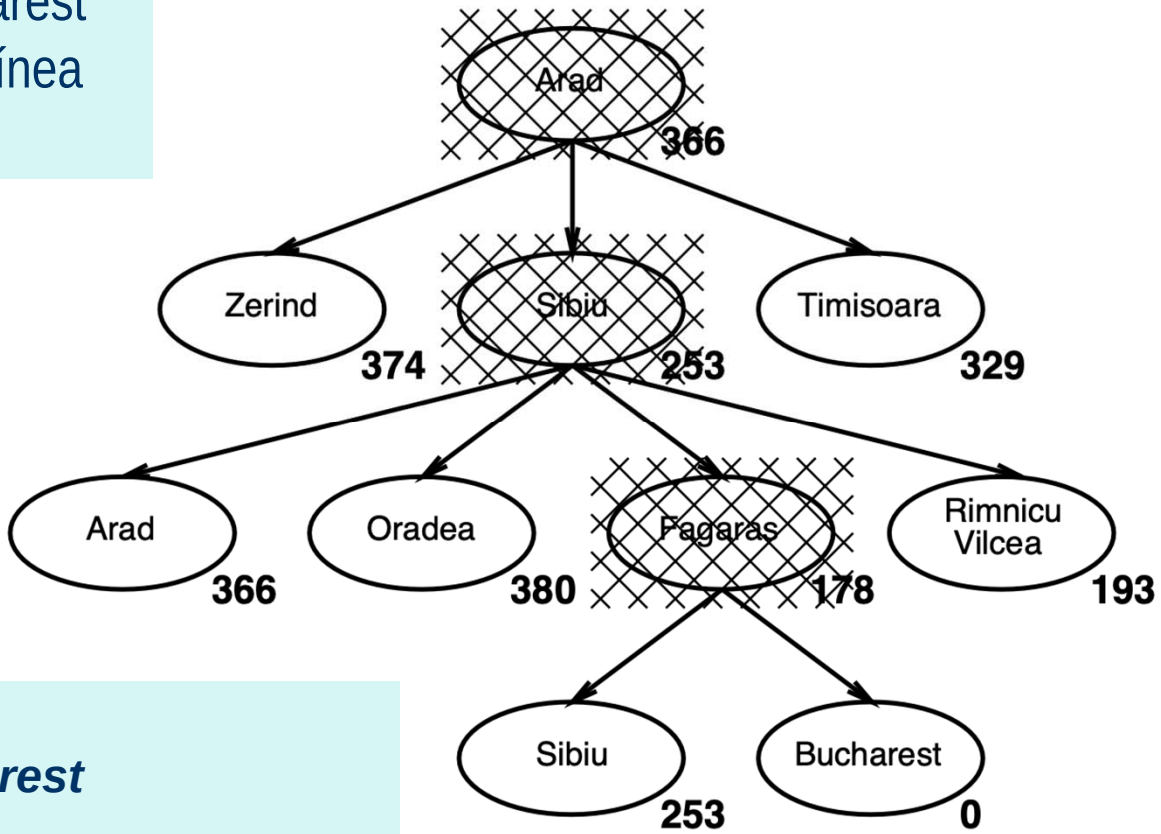
h(n) realiza una estimación del camino más barato a Bucarest desde Arad: distancia en línea recta



Solución: Arad, Sibiu, Fagaras, Bucarest

Total recorrido=140+99+211 = 450

h(n) realiza una estimación del camino más barato a Bucarest desde Arad: distancia en línea recta



Solución:
Arad, Sibiu, Fagaras, Bucarest

Coste real=*140+99+211* = ***450***

2.1 Búsqueda Voraz
Rendimiento

¿COMPLETA?	¿ÓPTIMA?	Complejidad en tiempo:	Complejidad en espacio:
• NO No garantiza encontrar una solución	• NO No garantiza encontrar la **mejor** solución	• O(bm) nº de nodos generados	• O(bm) longitud máxima Abiertos

(b: factor de ramificación,
m: profundidad máxima
d: profundidad de la solución óptima)

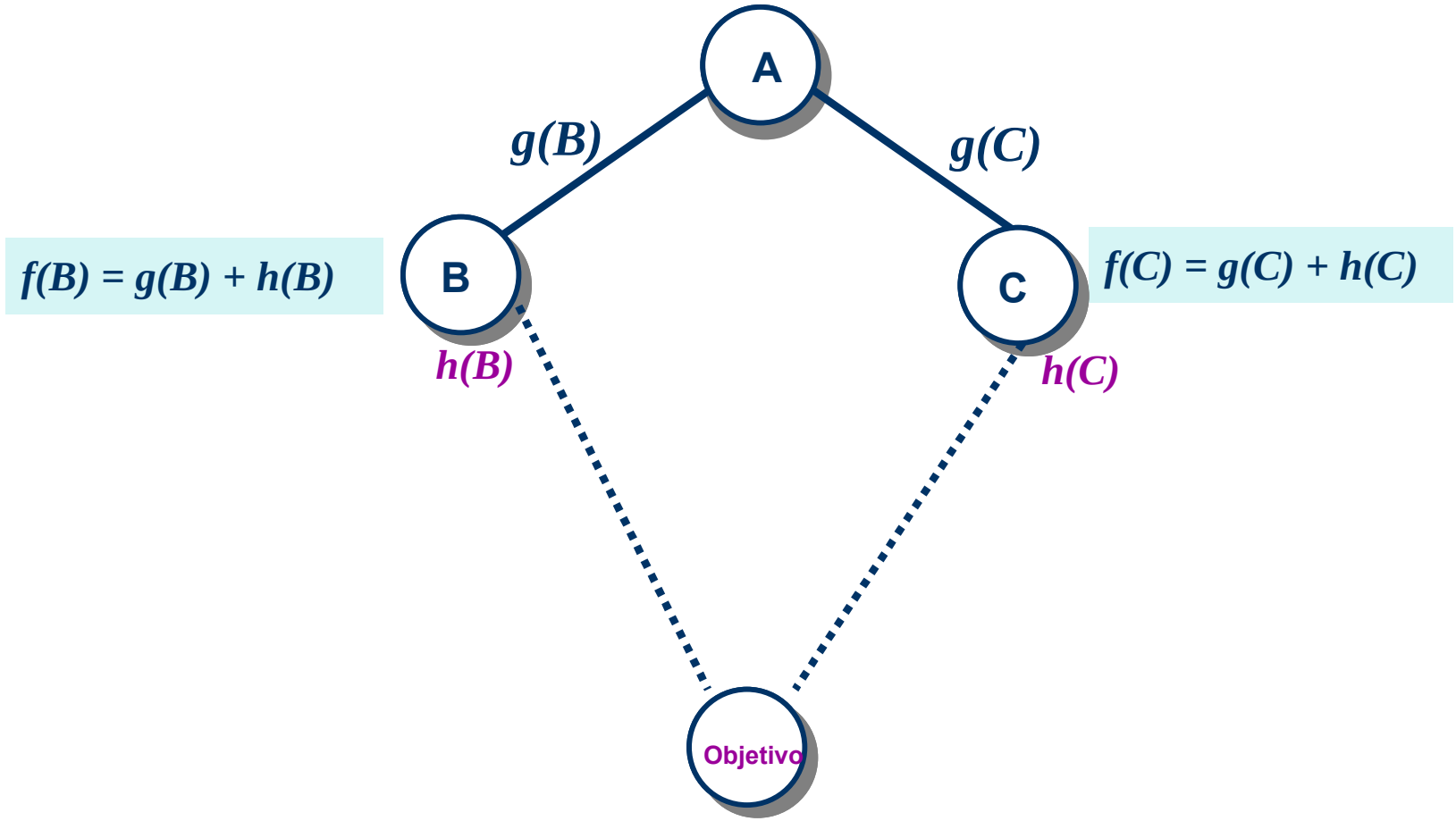
2.2 Algoritmo A*

- Trata de minimizar el coste estimado total de la solución.
- Para ello la función de evaluación f(n) calcula el coste menor estimado de una solución que pase por el nodo n:

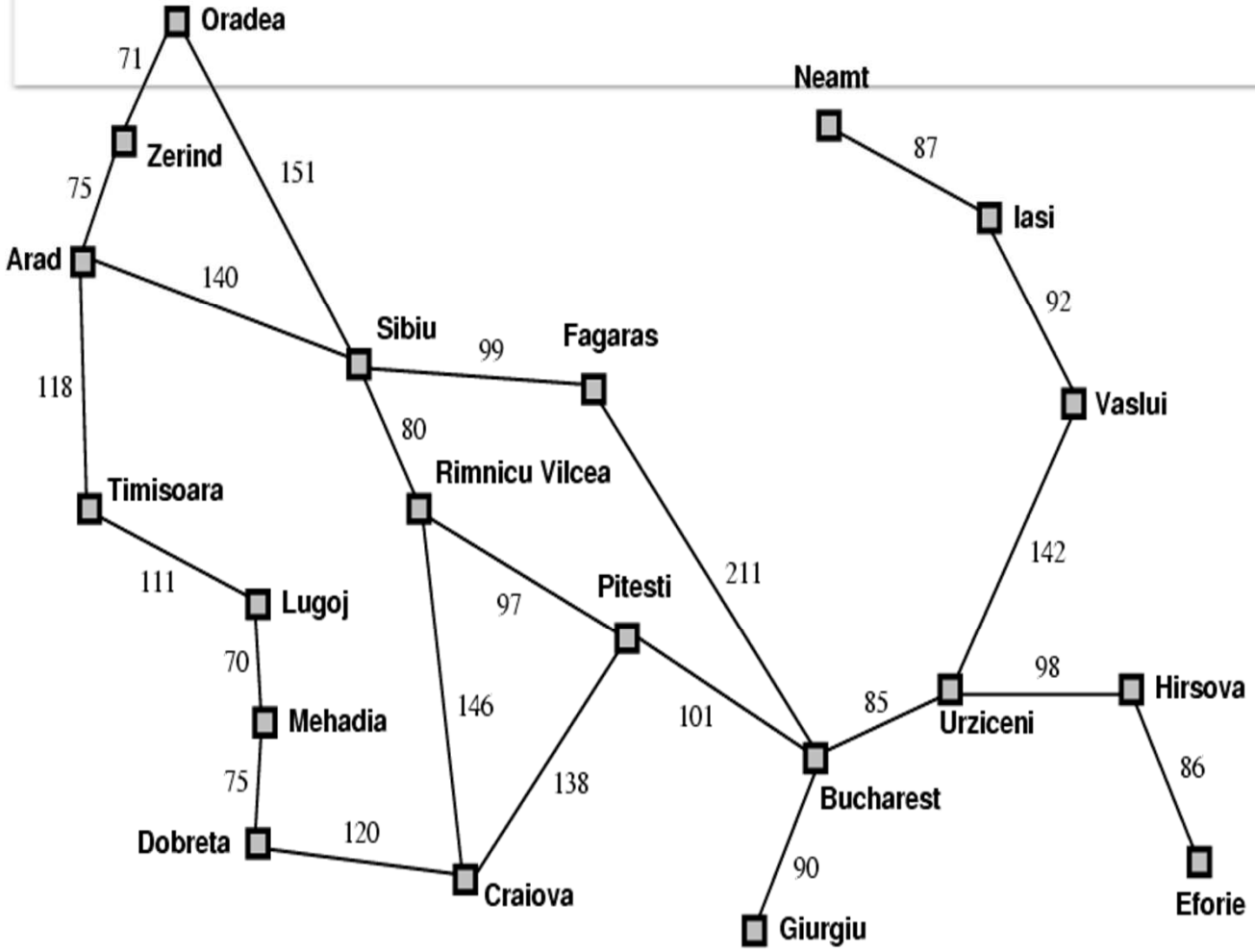
$$f(n) = g(n)+h(n)$$

  - g(n): coste de recorrer el camino desde el estado inicial hasta n
  - h(n): coste estimado de ir del estado n hasta el objetivo

2.2 Algoritmo A*
Función de evaluación en A*



Distancia entre ciudades rumanas



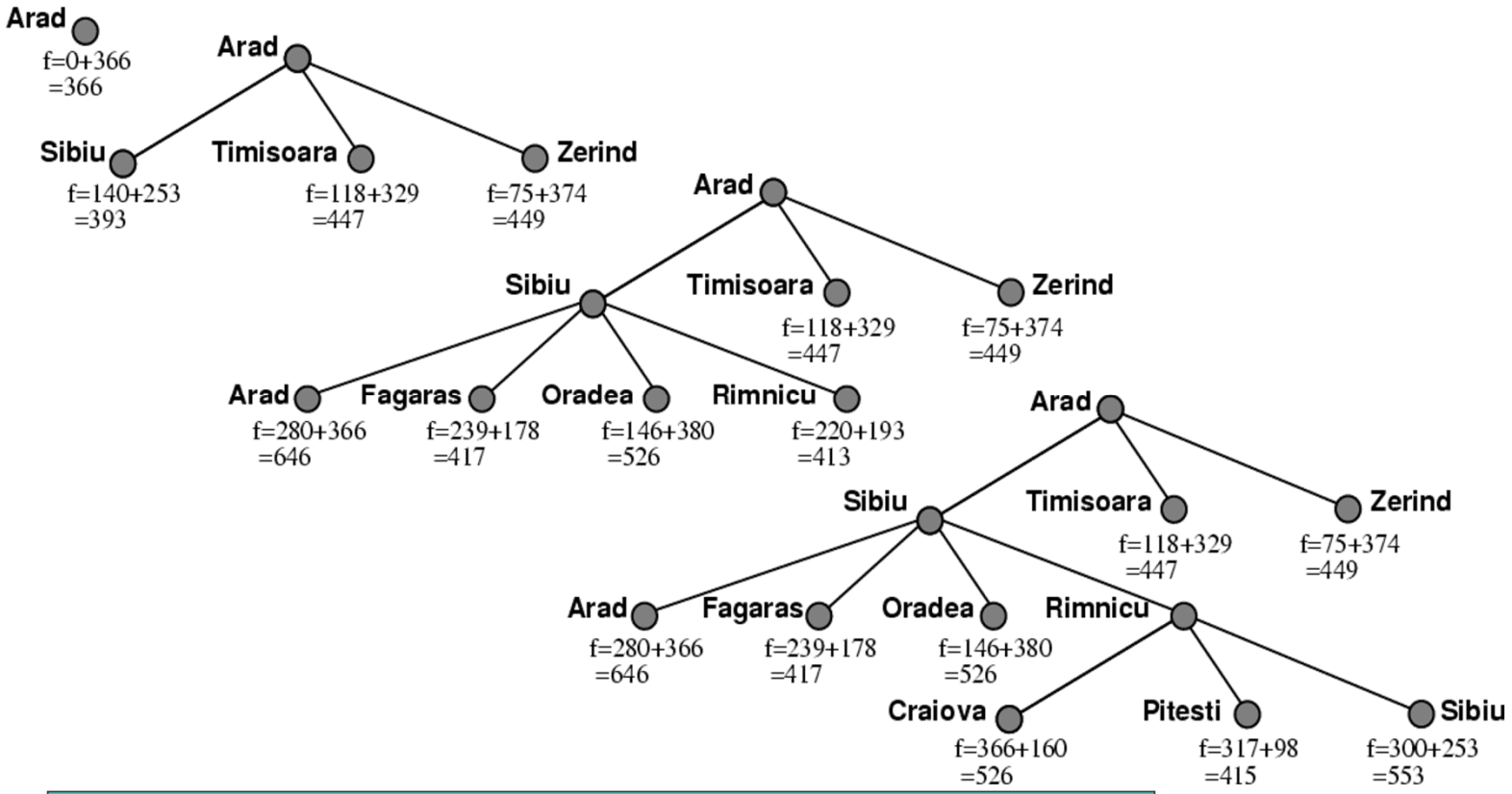
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Encontrar el camino más corto por carretera desde Arad a Bucarest

h(n): distancia en línea recta de la ciudad *n* a Bucarest

g(n) : distancia desde Arad a la ciudad *n*

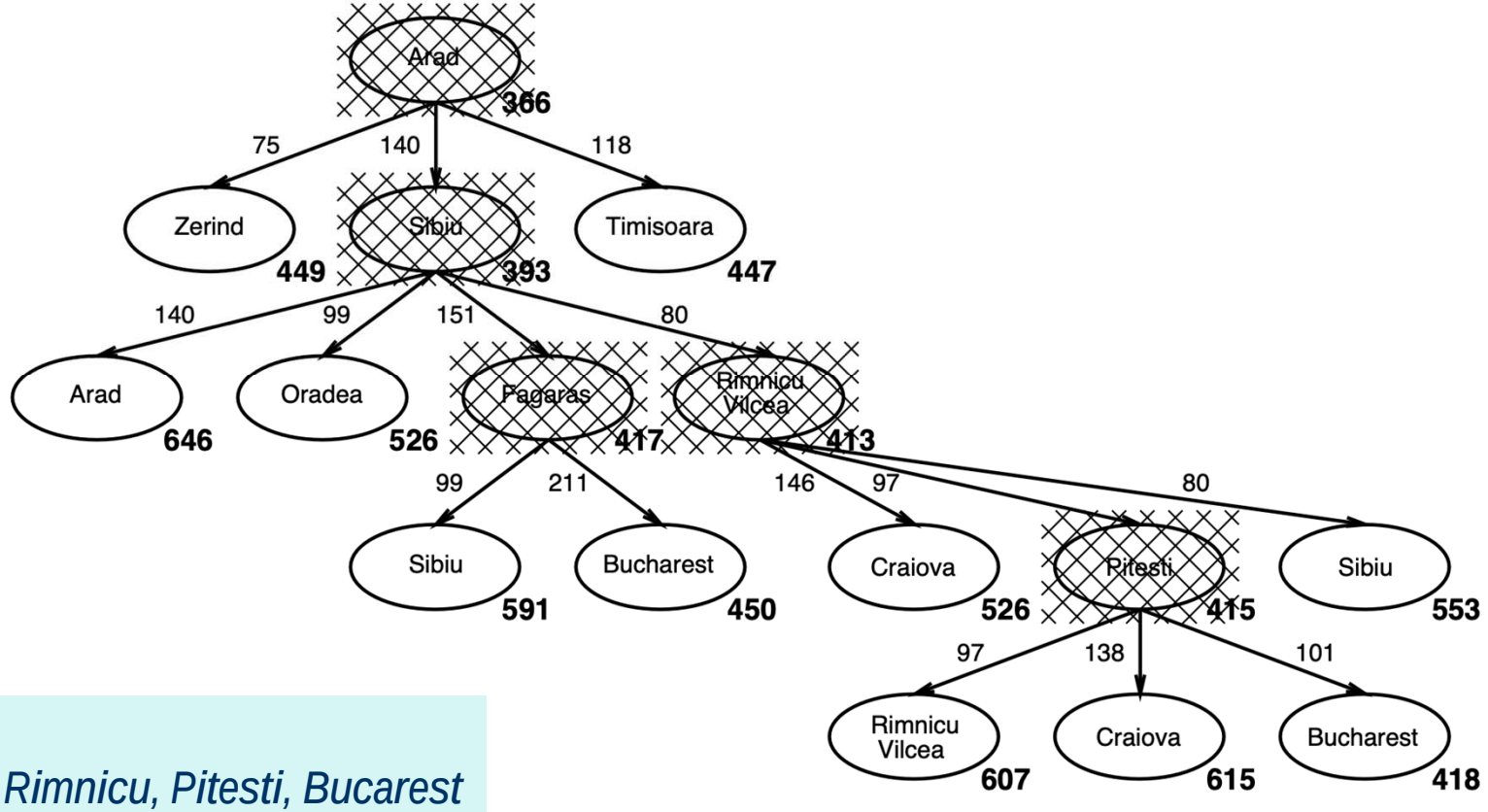
h(n) : distancia en línea recta desde *n* a Bucarest



Total Recorrido = 140 + 80 + 97 + 101 = **428**

¡ES MEJOR !

2.2 Algoritmo A*
Ejemplo A*



Solución:
Arad, Sibiu, Rimnicu, Pitesti, Bucarest

Coste real= 140 + 80 + 97 + 101 = 418

¡ES MEJOR !

2.2 Algoritmo A*
Rendimiento

¿COMPLETA?	¿ÓPTIMA?	Complejidad en tiempo	Complejidad en espacio
• cuando elimina los estados repetidos	• cuando h es **admisible** o consistente	• O(bm) • Si h es **admisible**: O(bd)	• O(bm) • Si h es **admisible**: O(bd)

2.3 Algoritmo General de Búsqueda
Control de los estados repetidos

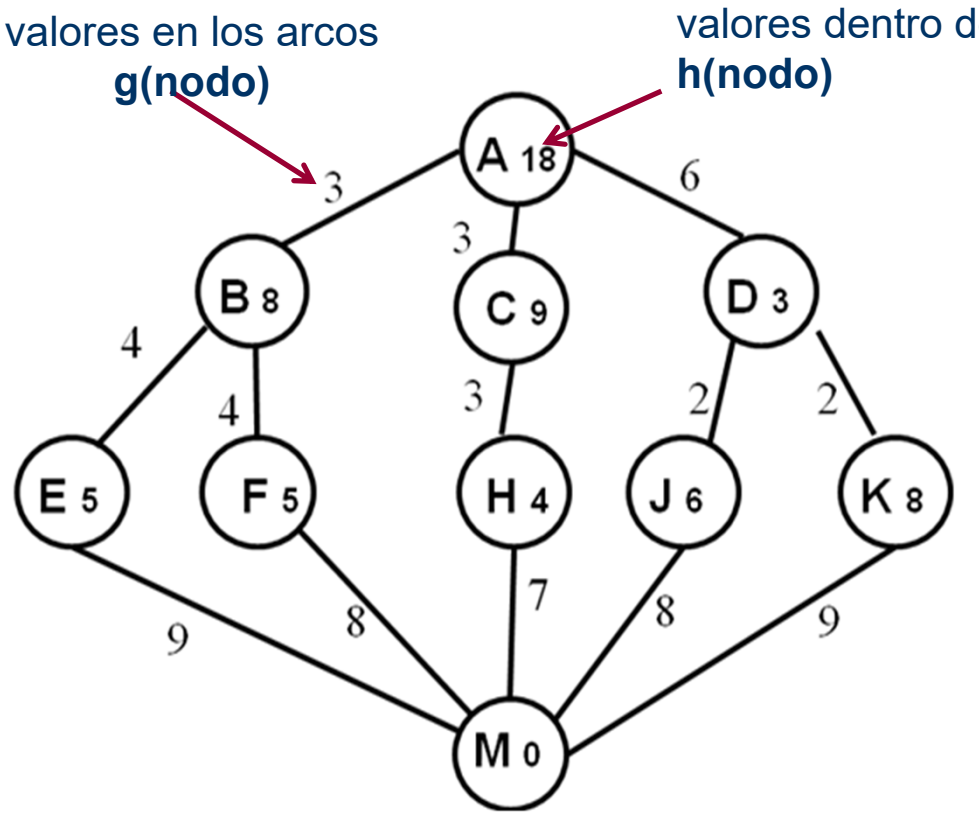
- Si Actual es un estado repetido que está en la lista de CERRADOS:
  - Si la nueva función de evaluación f(Actual) es menor que la del nodo en Cerrados, se considera este nodo para su expansión
  - Si no (el valor de f(Actual) es mayor o igual) no se considera este nodo para su expansión ni se guarda en ninguna lista
  - ¡Atención! No hacemos nada con sus sucesores; ya se reabrirán si hace falta.

2.3 Algoritmo General de Búsqueda

```
Solucion: función Búsqueda (tNodo: Inicial, entero: estrategia)
inicio
   tNodo Actual
   tLista: Abiertos ← {Inicial}  // El nodo inicial se guarda en Abiertos
   logico Objetivo: Falso
   mientras (No Vacia(Abiertos)) Y (No Objetivo)
                              Actual ← Primero(Abiertos) // selecciona primer nodo de Abiertos
                              Objetivo ← EsObjetivo(Actual)
                              si NO (Objetivo)  entonces
                                 si NO(Repetido) entonces
                                     Sucesores ← Expandir(Actual)  //calcula heurística a cada sucesor
                                     Abiertos ← Ordena {Abiertos+Sucesores} //en orden creciente de f(n)
                              fin_si
                              Cerrados ←  {Cerrados+Actual}
  fin_mientras
  si Objetivo entonces
         devolver  Camino a la Solución
  si_no devolver Fallo
fin_función
```

Ejercicio Genérico

Aplica Estrategias de búsqueda Voraz y A*



2.4. Mejoras del Algoritmo A*

- La principal desventaja de A* es que mantiene todos los nodos generados en memoria (crecimiento exponencial)

- ALTERNATIVAS:
  - Profundidad Iterativa
  - Memoria Acotada

2.4. Mejoras del Algoritmo A*
A* de Profundidad Iterativa

- Utiliza como criterio de corte f-valor:
  - Expandir nodo sólo si f(nodo) <= f-valor
  - Actualizar f-valor = Mín{ f(nodo)>f-valor }
    - mínimo valor que supere el límite establecido en la iteración anterior
- Sufre una regeneración excesiva de nodos.

2.4. Mejoras del Algoritmo A*
A* con Memoria Acotada Simplificada

- Avanza como A* hasta que la memoria esté llena.
- A*MS retira el peor nodo hoja (mayor f-valor) y expande la mejor hoja.
- Completo si d es menor que el tamaño de la memoria.
- Óptimo si la solución óptima es alcanzable.
- No es eficiente en problemas grandes porque la limitación de memoria puede hacer que un problema sea intratable desde el punto de vista de tiempo de cálculo.

3. Funciones Heurísticas
Admisibilidad

Admisible: Una heurística h(n) admisible es una función que nunca sobrestima el coste real de alcanzar el estado final.

- *La heurística distancia en línea recta es admisible*

Óptima: Si la función heurística h(n) es admisible, A* encontrará el camino de menor coste hacia la solución.

A* es óptima si h(n) es admisible (con árboles de búsqueda): porque f(n)=g(n)+h(n) nunca sobrestima el coste actual de la mejor solución hacia n.

3. Funciones Heurísticas
Demostración A* es Óptimo (Árb. Búsq.)

G1: objetivo óptimo

$$f(G1)=g(G1)+h(G1)=g(G1)+0=C^*$$

G2: objetivo subóptimo

$$f(G2)=g(G2)+h(G2)=g(G2)+0> C^*$$

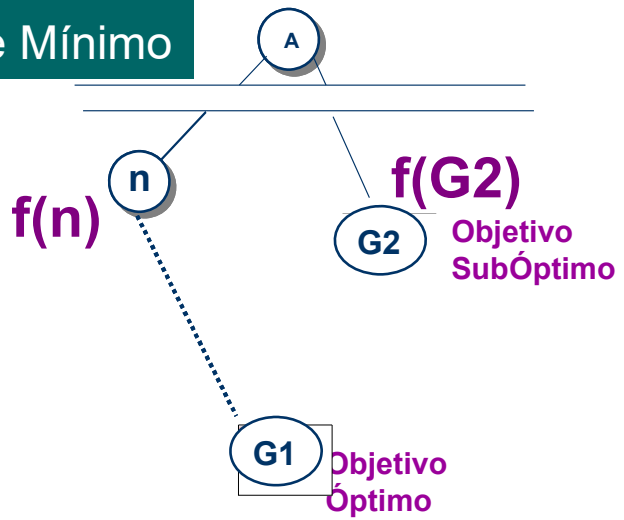
h(G2) = 0
g(G2) MAYOR que C*

n es un nodo en el camino de la solución óptima

$$f(n)=g(n)+h(n) <= C^*$$

h admisible

h(G1) =**0**
C* Coste Mínimo



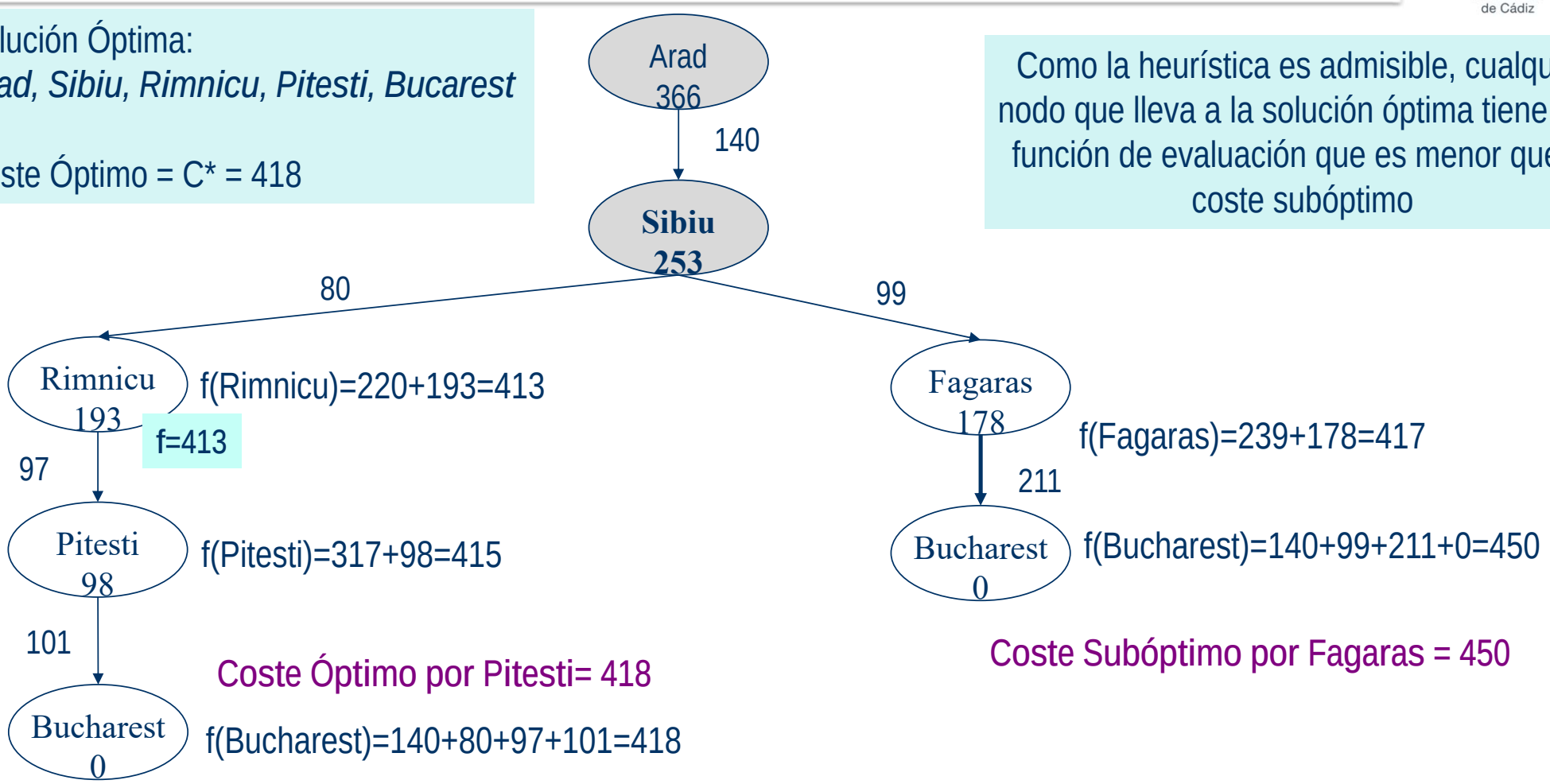
f(n) MENOR O IGUAL que C*

A* nunca seleccionará el nodo G2 porque
f(n) ≤ C* < f(G2)

3. Funciones Heurísticas
Admisibilidad

Solución Óptima:
Arad, Sibiu, Rimnicu, Pitesti, Bucarest

Coste Óptimo = C* = 418



Como la heurística es admisible, cualquier nodo que lleva a la solución óptima tiene una función de evaluación que es menor que el coste subóptimo

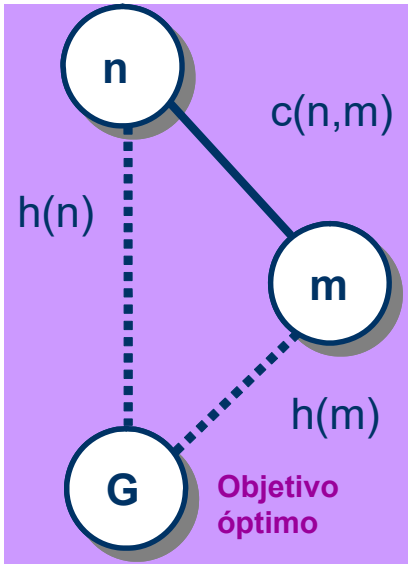
Coste Óptimo por Pitesti= 418

Coste Subóptimo por Fagaras = 450

3. Funciones Heurísticas
Consistencia o monotonía

Consistente: Una heurística *h(n)* es consistente si para cada nodo *n* y para cada sucesor *m* de *n* se cumple:

- h(n) <= c(n,m)+h(m), $\forall$ (n,m)

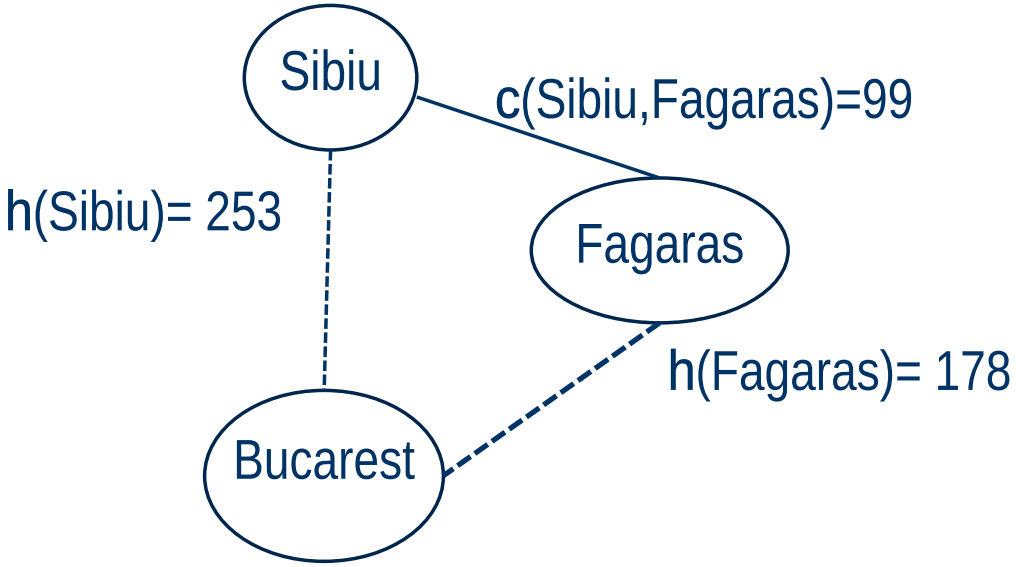


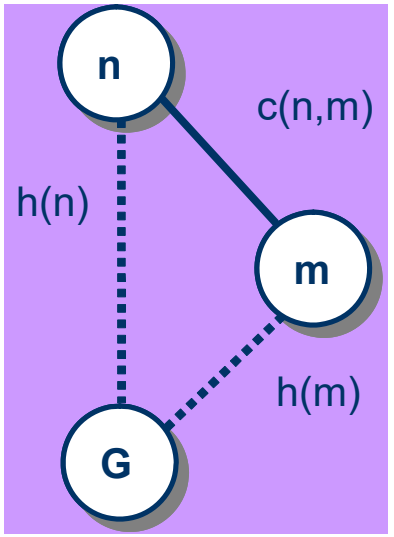
Óptima: si h(n) es consistente (con grafos de búsqueda) porque se cumple esa desigualdad triangular. A* encuentra el camino óptimo.

Si h(n) es consistente también es admisible.

3. Funciones Heurísticas
Consistencia en el ejemplo de las ciudades rumanas

- Heurística: distancia en línea recta





h(Sibiu) <= c(Sibiu,Fagaras) + h(Fagaras)

253<=277

3. Funciones Heurísticas
Diseño de Funciones Heurísticas

Las funciones heurísticas son específicas del dominio del problema.

¿Cómo diseñar heurísticas eficientes?

Valores no negativos
El mejor es el menor
Los objetivos tienen heurística 0

- Relajación de precondiciones:
  - 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.

3. Funciones Heurísticas
Diseño de Funciones Heurísticas

Ejemplo del 8-puzle:

Problema real:

- Una ficha A puede intercambiarse con B (o moverse a la posición B) si:
  - Las posiciones de A y B son adyacentes
  - B es el hueco

Problema relajado:

- No se necesita adyacencia para mover una ficha A a la posición de B
- No es necesario que B sea un hueco

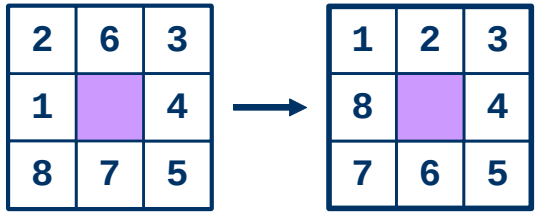
h2 = Σdistancias de Manhattan

h1 = nº de piezas mal colocadas

3. Funciones Heurísticas
Ejemplo: Heurísticas para el 8-puzle

- h1 = nº de piezas mal colocadas

$$h1(x) = \text{fichas}(1,2,6,7,8)=5$$



- h2 = suma de las distancias de Manhattan de las posiciones a sus objetivos.
  - La distancia de Manhattan es el nº de filas y columnas que restan de la posición actual de una pieza a su posición final. Por ejemplo, la distancia Manhattan de la pieza 2 sería de 1, de la pieza 5 sería 0, etc.

$$h2(x) = 1+1+0+0+0+2+1+1=6$$

Ambas son admisibles
¿Cuál es mejor?

3. Funciones Heurísticas
Dominancia

- Funciones Heurísticas Dominantes:
  - h2 domina a h1 (ambas admisibles) si
    - $\forall n, h2(n) \geq h1(n)$
  - h2 nunca generará más nodos que h1

- Una heurística dominante expande menos nodos
  - La distancia de Manhattan esta más informada que el número de piezas mal colocadas.

3. Funciones Heurísticas
Eficiencia en las Funciones Heurísticas

- **Factor de Ramificación Efectivo** b* que debería tener un árbol equilibrado de profundidad d para contener N+1 nodos:

$$N + 1 = 1 + b^* + b^{*2} \ldots + b^{*d}$$

➡ Mejor heurística cuanto más cercano a 1 sea b*

- Una heurística bien diseñada debe tender a valores de b* cercanos 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

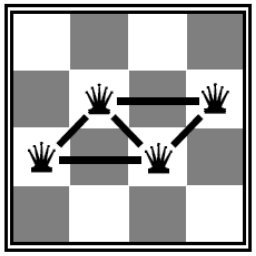
3. Funciones Heurísticas
Combinación

- $h(n)=\max\{(h_1(n), h_2(n), \ldots, h_p(n)\}$, siendo $h_1, h_2, \ldots, h_p$ admisibles.
- Aprender h(n) mediante la solución de muchos problemas (nodo, costo)
- Combinación de características :

$$h(n) = c_1h_1(n) + c_2h_2(n)$$

4. Búsqueda Local

- Cuando el camino a la solución es irrelevante, sólo interesa el estado objetivo:
  - Guardan sólo un estado en memoria: el estado actual.
  - Se mueven a los nodos vecinos del nodo actual.



- Bucle que continuamente se mueve en la dirección de un valor:
  - Se generan los sucesores de un estado *n*, y se devuelve el sucesor *m*, por ser el que tiene **mejor valor** de la función de evaluación:

$$f(m) < f(n) \text{ (decreciente)}$$

Algoritmo B. en Escalada

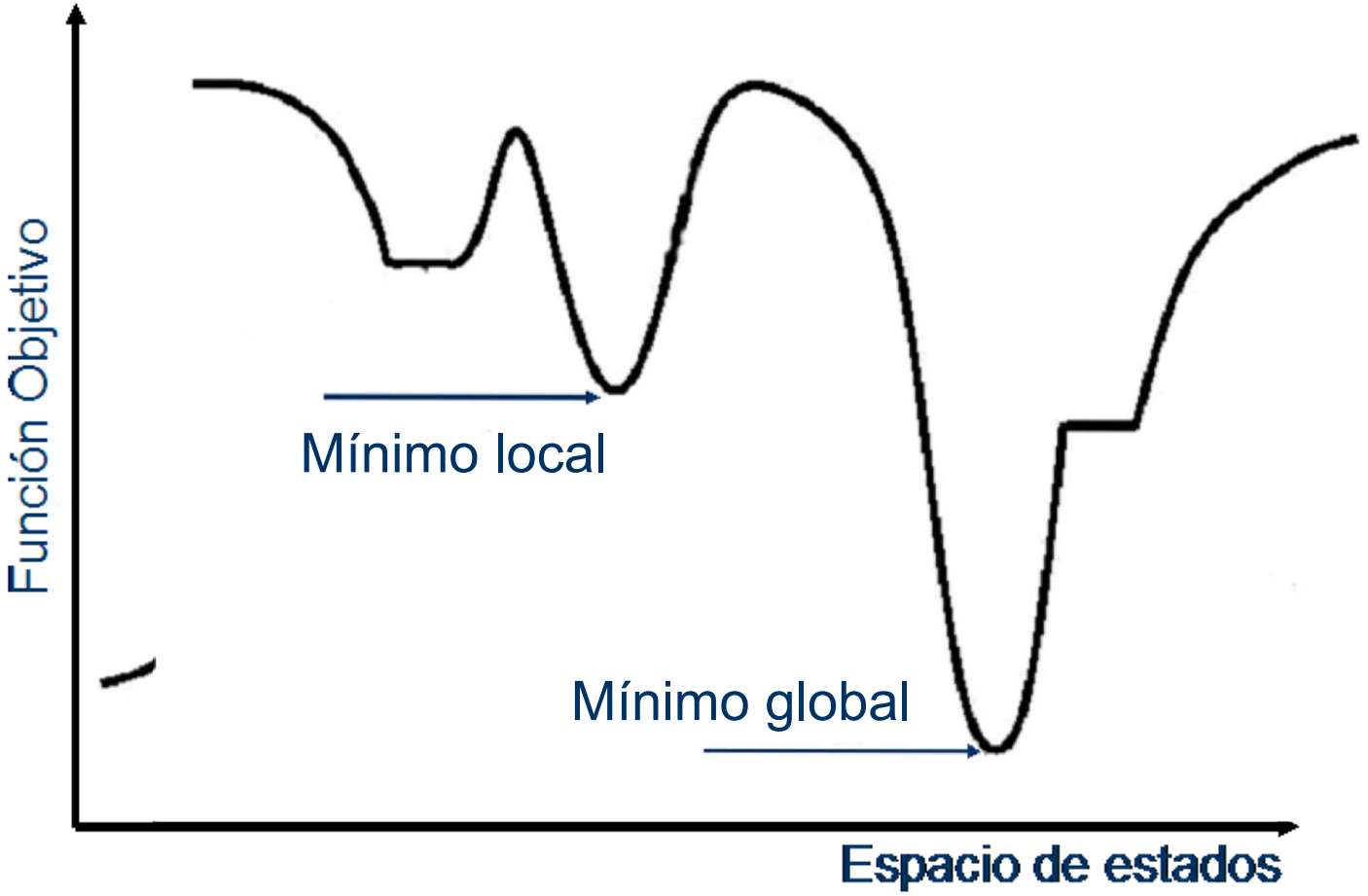
La lista de ABIERTOS sólo mantendría un único estado después de aplicar los operadores a actual

```
vecino:=sucesor de actual con f menor
si f(vecino) < f(actual) (DECRECIENTE)
      entonces ABIERTOS:= vecino
fin_si
```

4.1 Búsqueda en Escalada

- Sigue el recorrido a través de los nodos en los que el valor de dicha función sea máximo (cuesta arriba) o mínimo (depende del planteamiento).
- No mantiene un árbol de búsqueda, tan sólo una estructura con el estado y el valor de la función objetivo.
- La Búsqueda en Escalada sólo mira a los vecinos inmediatos al estado actual.
- Termina cuando alcanza un extremo (máximo o mínimo) donde ningún vecino tiene un valor mejor.

4.1 Búsqueda en Escalada
Problemas



41. Búsqueda en Escalada
Rendimiento

- No es completa
- No es óptima
  - Pero puede encontrar soluciones aceptables
- Complejidad en tiempo:
  - nº de nodos generados O(b·m)
- Complejidad en espacio:
  - Almacena un estado O(1)

- Utilizan poca memoria.
- Pueden encontrar soluciones razonables en espacios de estados grandes o infinitos.
- Pueden quedar atrapados en máximos/mínimos locales.
- No son sistemáticos en la búsqueda.

4.2 Búsqueda por Haz Local

- Guarda la pista de **k estados.**
- Comienza con estados generados aleatoriamente
- En cada paso, se generan todos los sucesores de los k estados.
- Si alguno es un objetivo, finaliza.
- Si no, se seleccionan los k mejores sucesores de la lista completa y se repite el proceso

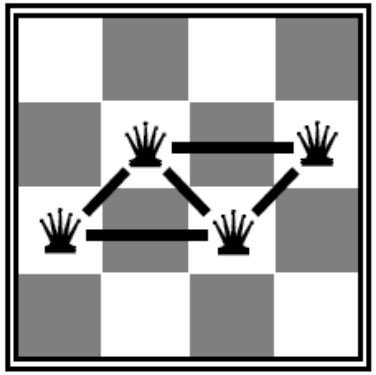
Rendimiento de Búsq. Haz Local

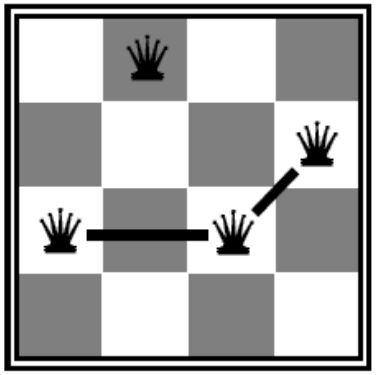
- Es completa
- No es óptima
  - Puede encontrar soluciones buenas.

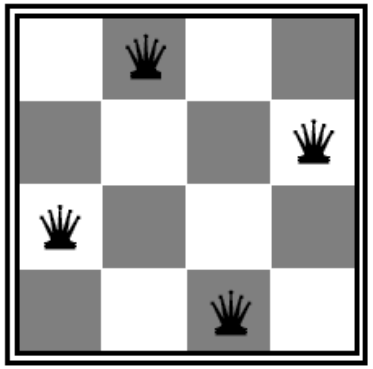
- Complejidad en tiempo:
  - nº de nodos generados $O(b \cdot m)$
- Complejidad en espacio:
  - Almacena k estados O(k)

El problema de las N-Reinas

Colocar n reinas sobre un tablero de n x n, sin que queden dos reinas en la misma columna, fila, o diagonal







Referencias

- Borrajo D. Et al. (1997), Inteligencia artificial : Métodos y técnicas. Editorial Centro de Estudios Ramón Areces. Madrid
- RUSSELL, S. y NORVIG, P.(2004) Inteligencia Artificial. Un enfoque moderno (2ª Ed.). Pearson Educación, S.A. Madrid.
- Fernández Galán y otros autores (203): Problemas Resueltos de Inteligencia Artificial Aplicada. Pearson.
- NILSSON N (2001): Inteligencia Artificial: Una nueva síntesis. McGrawHill.
- RICH, E. and KNIGHT, K., (1994) Inteligencia artificial. McGraw-Hill. Edición original: Artificial Intelligence.
- WINSTON, P. H. (1994), Inteligencia Artificial. Tercera Edición, p. xxv+805, Addison-Wesley Iberoamericana, Wilmington, Delaware, EE.UU