

Tema 4

Técnicas de Búsqueda Heurística

Inteligencia Artificial
2º curso Grado en Ingeniería Informática
Elisa Guerrero Vázquez
Esther L. Silva Ramírez

Técnicas de Búsqueda Heurística

1. BÚSQUEDA PRIMERO EL MEJOR (Best-First Search)

- Búsqueda Voraz o Avara (Greedy search)
- Algoritmo A*
- Mejoras al Algoritmo A*

2. FUNCIONES HEURÍSTICAS

- Propiedades

3. BÚSQUEDA LOCAL

- Búsqueda en escalada o Gradiente (Hill-climbing, Gradient descent)
- Haz Local (Beam search)

Objetivos

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

Introducción

Función Heurística $h(n)$

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 $h(n)$

- **Estima** el coste de elegir una ruta para llegar al objetivo. Dado un nodo n estima el coste de llegar desde n al nodo objetivo, siendo $h(\text{objetivo}) = 0$.
- Utiliza información del dominio específico del problema.
- No garantiza el éxito, pero suele ser mejor que la búsqueda a ciegas.

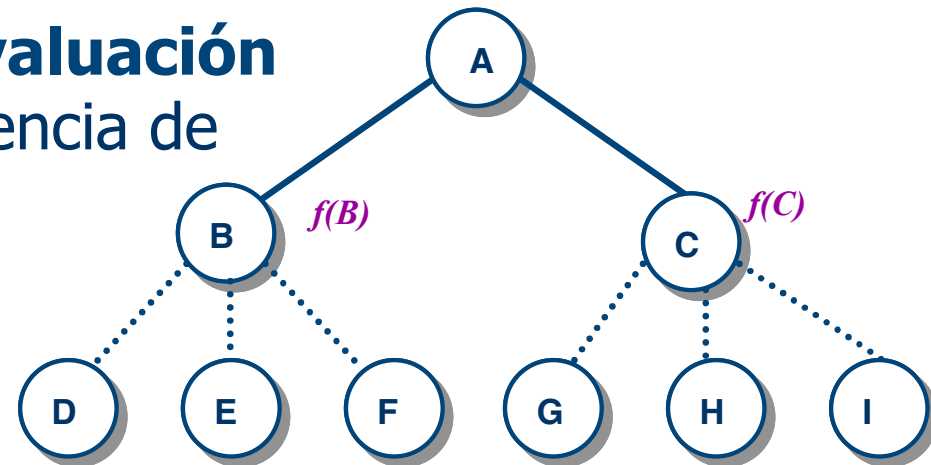
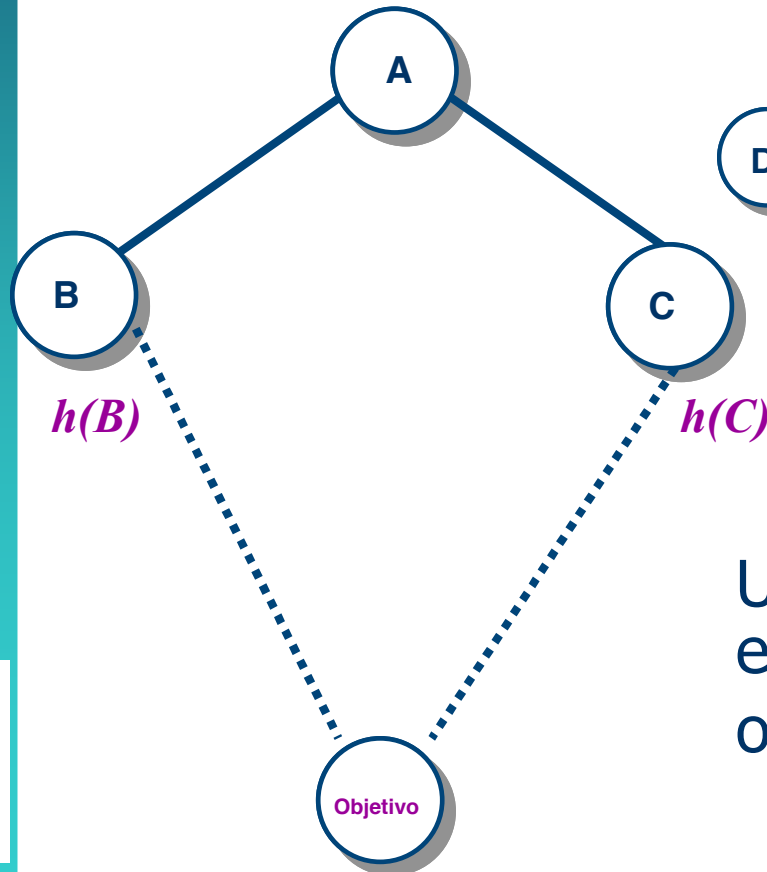
Introducción

Función de Evaluación $f(n)$

- 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
- Normalmente se establece el criterio de elegir el nodo con $f(n)$ de menor valor

Función de Evaluación $f(n)$ y Función Heurística $h(n)$

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



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

Introducción

Función de Evaluación $f(n)$

- La función de evaluación $f(n)$ puede tener dos componentes:
 - Coste del camino $g(n)$: coste del camino desde el nodo inicial al nodo n .
 - Función heurística $h(n)$: estimación del coste del camino desde el nodo n al nodo objetivo.

Si n es nodo objetivo $h(n) = 0$

- La función de evaluación $f(n)$ puede estar definida:
 - $f(n) = h(n)$
 - $f(n) = g(n) + h(n)$

Técnicas de Búsqueda Heurística

1. BÚSQUEDA PRIMERO EL MEJOR (Best-First Search)

- Búsqueda Voraz o Avara (Greedy search)
- Algoritmo A*
- Mejoras al Algoritmo A*

2. FUNCIONES HEURÍSTICAS

- Propiedades

3. BÚSQUEDA LOCAL

- Búsqueda en escalada o Gradiente (Hill-climbing, Gradient descent)
- Haz Local (Beam search)

Búsquedas Primero el Mejor (Avara y A*)

- Buscan el nodo que **parece** ser el mejor (según la estimación que se obtiene con la función de evaluación).
- Ordenan ascendentemente la lista ABIERTOS según el valor de la función $f(n)$ asociada 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

Búsquedas Primero el Mejor (Avara y A*)

- Greedy Search (Búsqueda Avara):

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

- Algoritmo A*

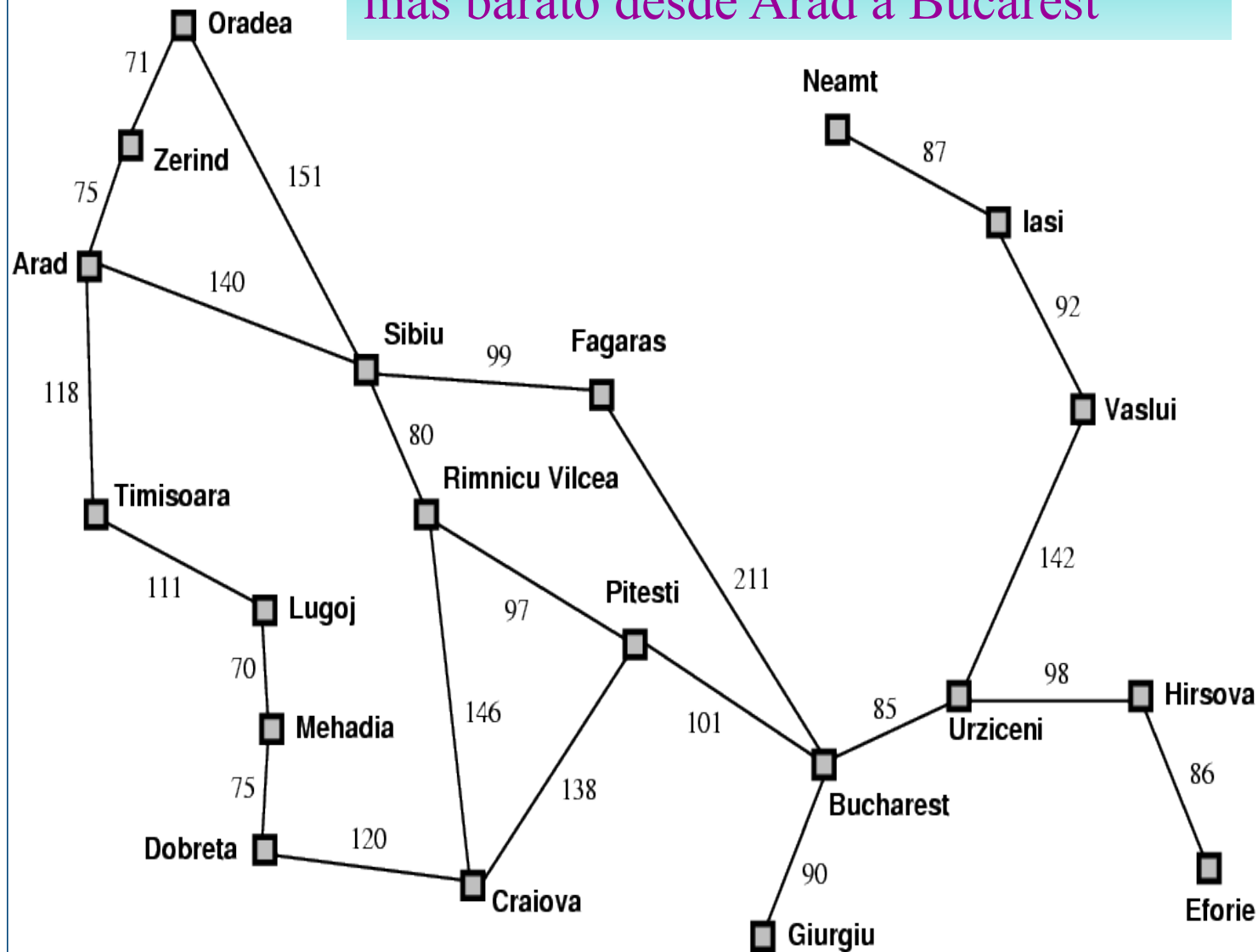
$$f(n) = g(n) + h(n)$$

Búsqueda Voraz (Avara)

Greedy Search

- 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 de llegar hasta n
- 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.

Realizar una estimación del camino más barato desde Arad a Bucarest



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

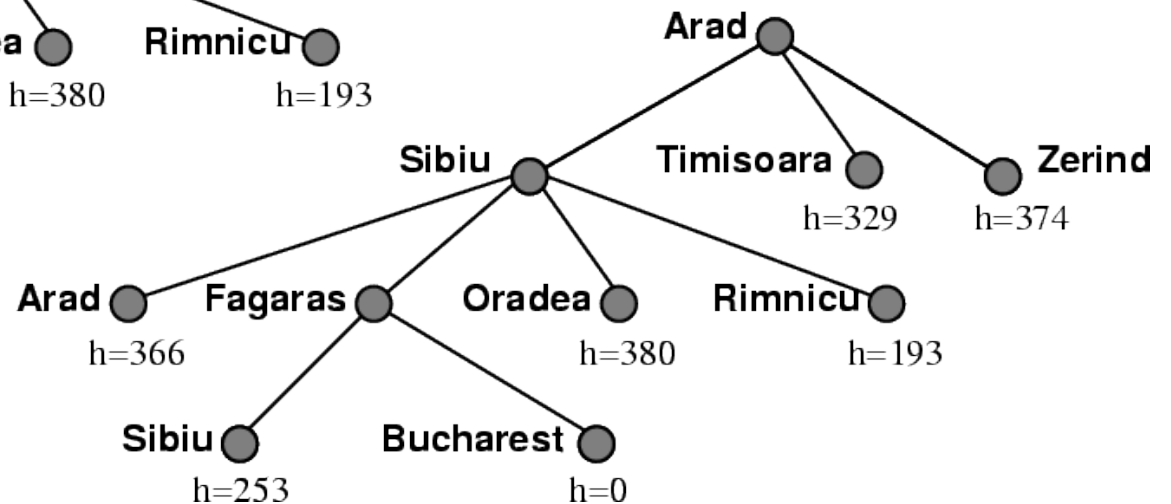
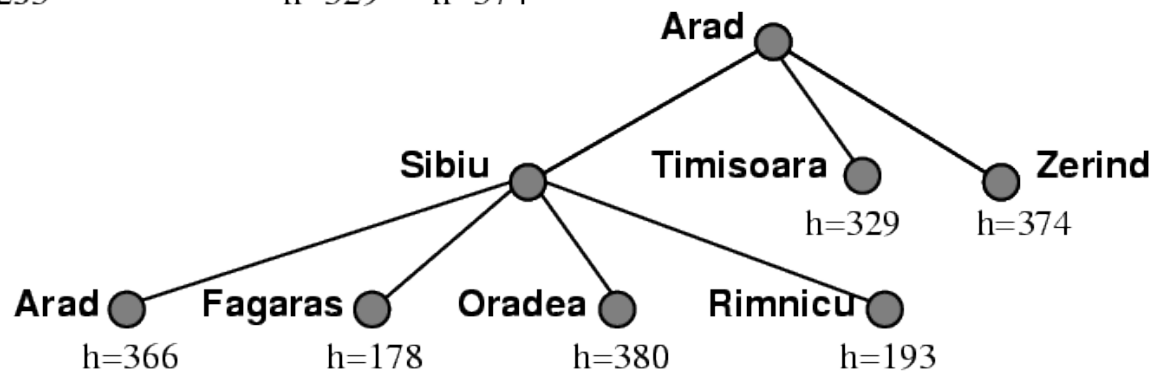
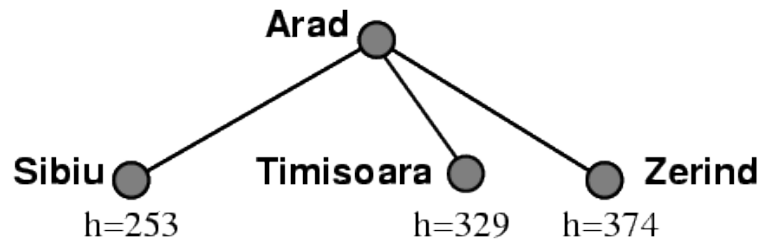
$h(n)$: distancia en línea recta de la ciudad n a Bucarest

$h(n)$ 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**

Arad
h=366



Rendimiento Búsqueda Voraz 1º el Mejor

- No es **completa**
- No es **óptima**
- **Complejidad en tiempo:**
nº de nodos generados $O(b^m)$
- **Complejidad en espacio:** longitud máxima que puede alcanzar la lista de estados almacenada en memoria $O(b^m)$

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

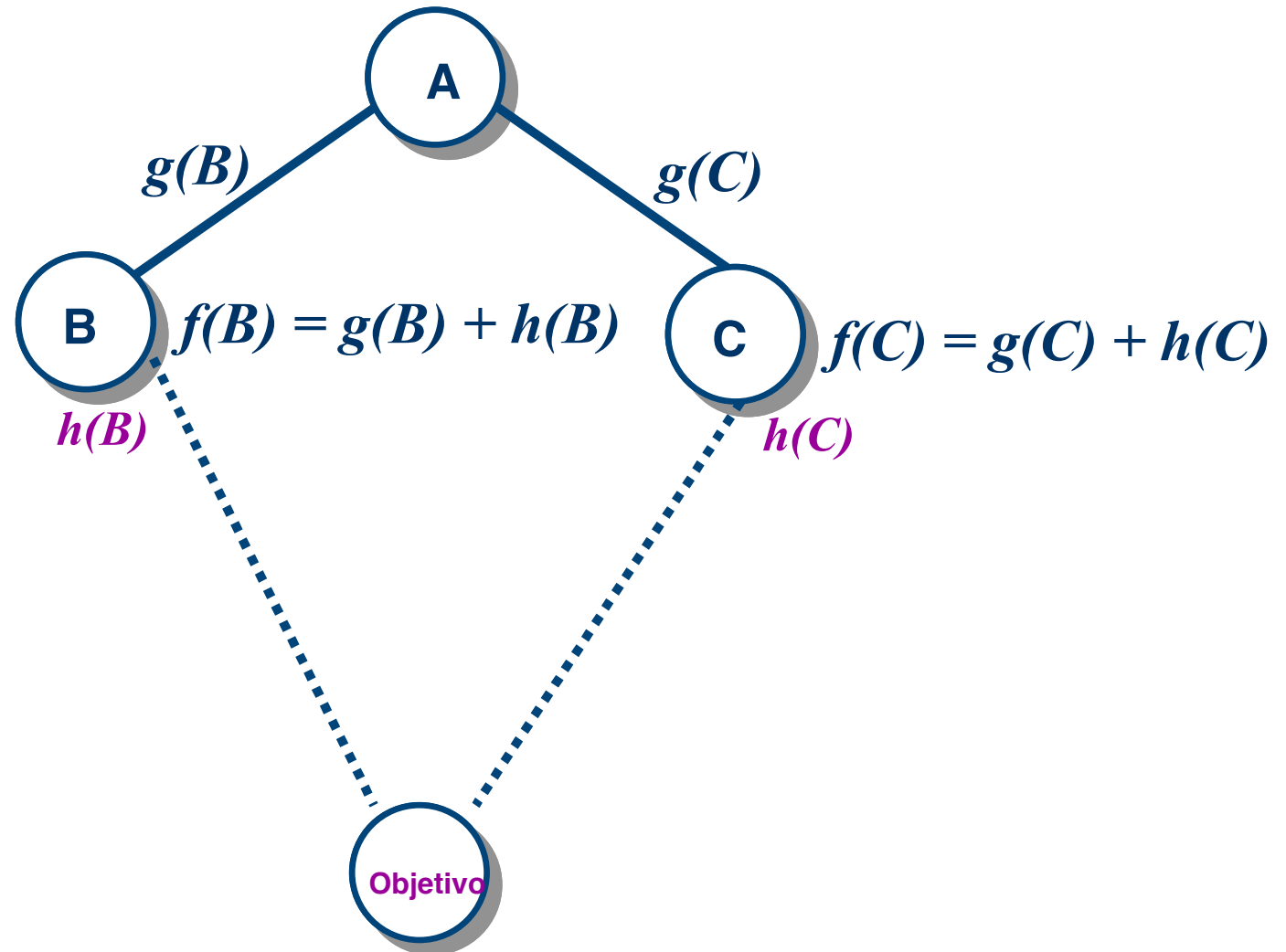
Algoritmo A*

- 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 :

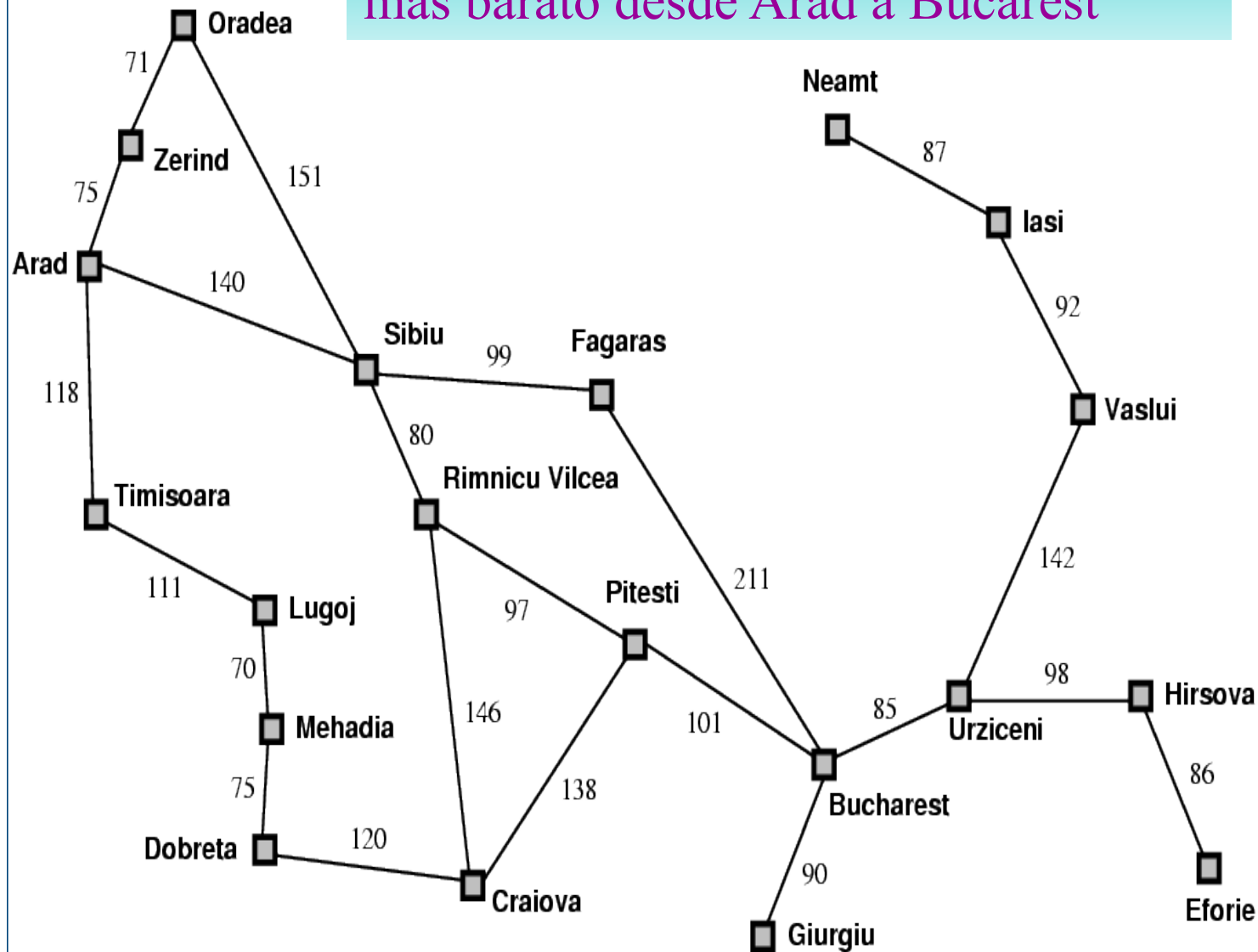
$$f(n) = g(n) + h(n)$$

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

Función de evaluación en A*



Realizar una estimación del camino más barato desde Arad a Bucarest

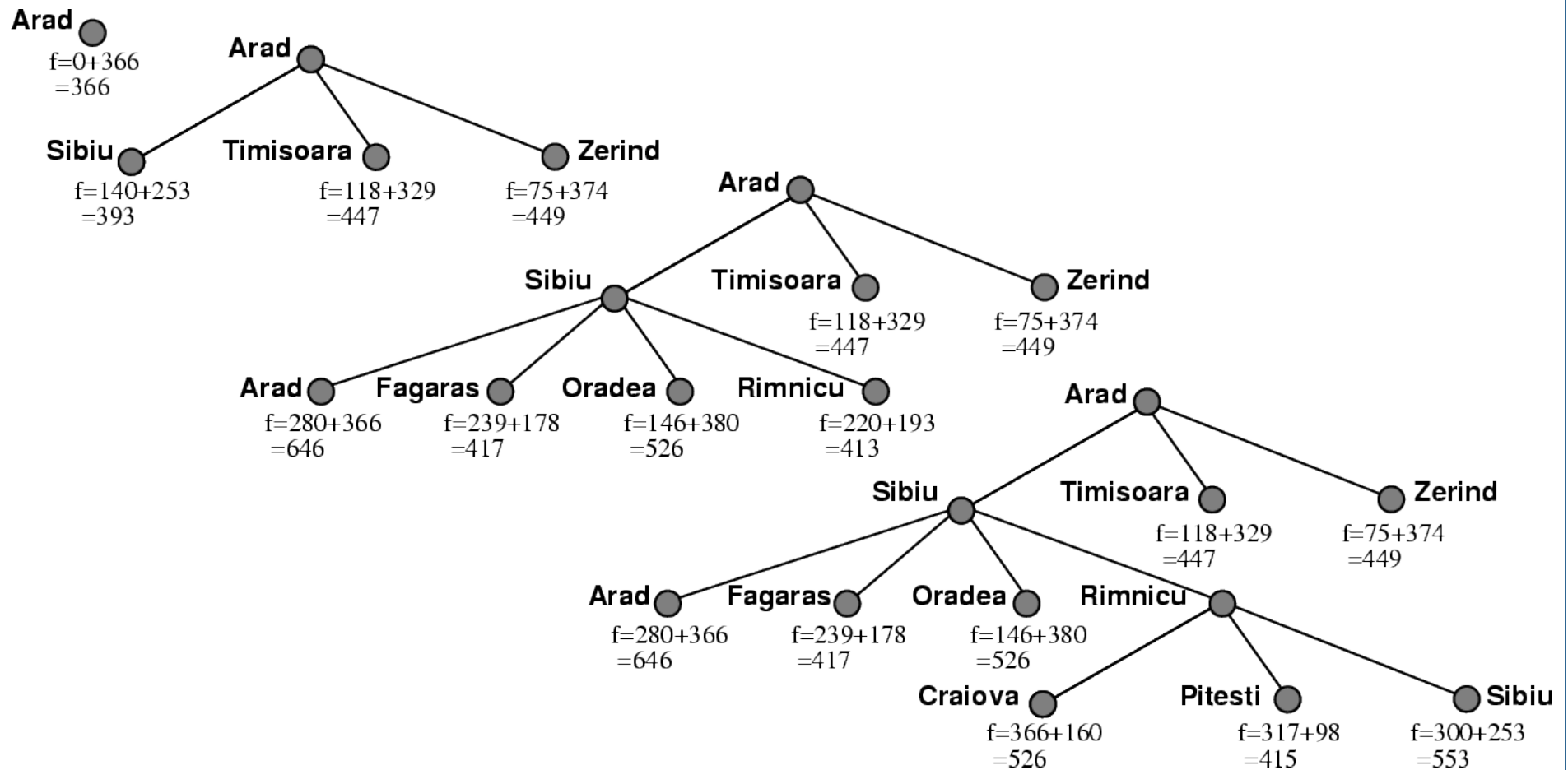


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

$h(n)$: 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 !

Rendimiento de A*

- **Completo:** cuando elimina los estados repetidos
- **Óptimo:** cuando ***h*** es **admisible** (*h* nunca sobreestima el coste real de alcanzar la meta) y **consistente** (desigualdades triangulares entre nodos sucesores)
- **Complejidad en tiempo:**
nº de nodos generados $O(b^m)$
Si ***h*** es **admisible:** $O(b^d)$
- **Complejidad en espacio:**
Longitud máxima que puede alcanzar la lista de estados almacenada en memoria $O(b^m)$
Si ***h*** es **admisible:** $O(b^d)$

Algoritmo General de Búsqueda

Solucion: función Búsqueda (tNodo: Inicial, entero: estrategia)

inicio

tNodo Actual

tLista: Abiertos \leftarrow {Inicial} // El nodo inicial se guarda en Abiertos

logico Objetivo: Falso

mientras (No Vacía(Abiertos)) **Y** (No Objetivo)

Actual \leftarrow Primero(Abiertos) // selecciona primer nodo de Abiertos

si EsObjetivo(Actual) **entonces**

Objetivo \leftarrow Verdadero

si_no

Sucesores \leftarrow Expandir(Actual) //calcula heurística a cada sucesor

Abiertos \leftarrow **Ordena** {Abiertos+Sucesores} //en orden creciente de $f(n)$

fin_si

Cerrados \leftarrow {Cerrados+Actual}

fin_mientras

si Objetivo **entonces**

devolver Camino a la Solución

si_no devolver Fallo

fin_función

Control de los estados repetidos

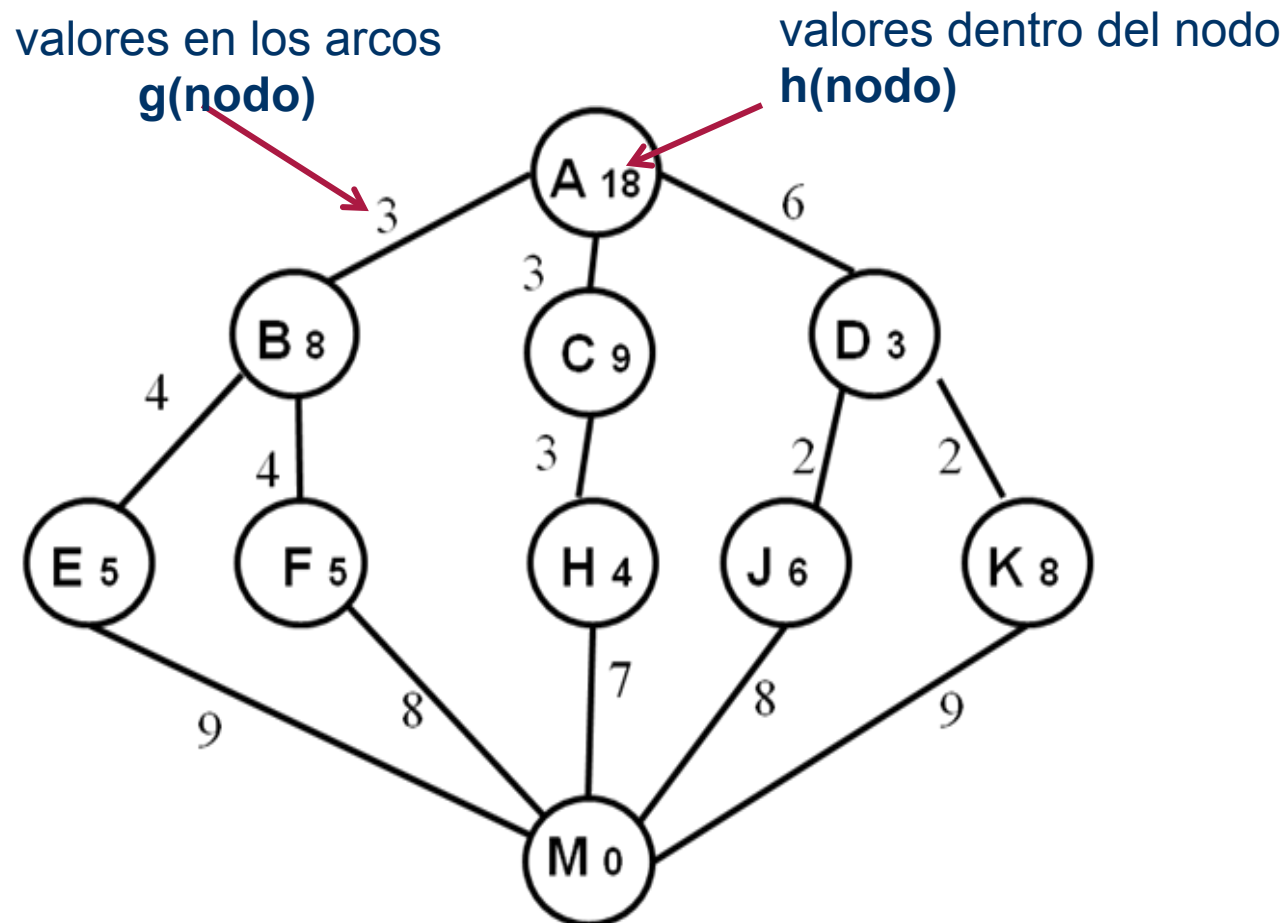
- Si *Actual* es un estado repetido que está en la lista de CERRADOS:
 - Si la nueva función de evaluación $f(Actual)$ es menor que la del nodo en Cerrados, se vuelve a insertar el nodo en Abiertos
 - Si no (el valor de $f(Actual)$ es mayor o igual) no se considera este nodo para su expansión ni se guarda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

Control de los estados repetidos

- Si *Actual* es un estado repetido que está en la lista de ABIERTOS:
 - Si la nueva función de evaluación $f(Actual)$ es menor que la del nodo en Abiertos, se actualiza el nodo al nuevo valor y se reordena la lista de Abiertos.
 - 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.**

Ejercicio Genérico

- Aplica Estrategias de búsqueda Voraz y A*



Mejoras del Algoritmo A*

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

A* de Profundidad Iterativa

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

Derivaciones del Algoritmo A*

A* BRPM: Búsqueda Recursiva 1º el Mejor

- Mantiene la pista del nodo con mejor función de evaluación que se encuentre en un camino alternativo disponible desde cualquier antepasado del nodo actual.
- Si el nodo actual excede el f-valor, vuelve al camino alternativo con mejor función de evaluación.
- Actualiza todos los nodos hacia atrás, y no descarta volver por un camino olvidado si la función de evaluación vuelve a ser mejor.
 - Regeneración excesiva de los nodos.
 - Óptimo si h es admisible
 - Complejidad en espacio: $O(b \cdot d)$

Derivaciones 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 función de evaluación) 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

Técnicas de Búsqueda Heurística

1. BÚSQUEDA PRIMERO EL MEJOR (Best-First Search)

- Búsqueda Voraz o Avara (Greedy search)
- Algoritmo A*
- Mejoras al Algoritmo A*

2. FUNCIONES HEURÍSTICAS

- Propiedades

3. BÚSQUEDA LOCAL

- Búsqueda en escalada o Gradiente (Hill-climbing, Gradient descent)
- Haz Local (Beam search)

Admisibilidad

- **Admisibilidad:** Una heurística $h(n)$ admisible es una función que nunca sobrestima el coste real de alcanzar el estado final
 - *P.ej. La heurística distancia en línea recta es admisible*
- **Óptimo:** Si la función heurística $h(n)$ es admisible, el algoritmo de búsqueda encontrará el camino más barato hacia la solución con el menor número de pasos
- **A* es óptimo 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

Demostración A* es Óptimo (Árb. Búsq.)

- G1: objetivo óptimo

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

$$h(G1) = 0$$

C* Coste Mínimo

- G2: objetivo subóptimo

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

$$h(G2) = 0$$

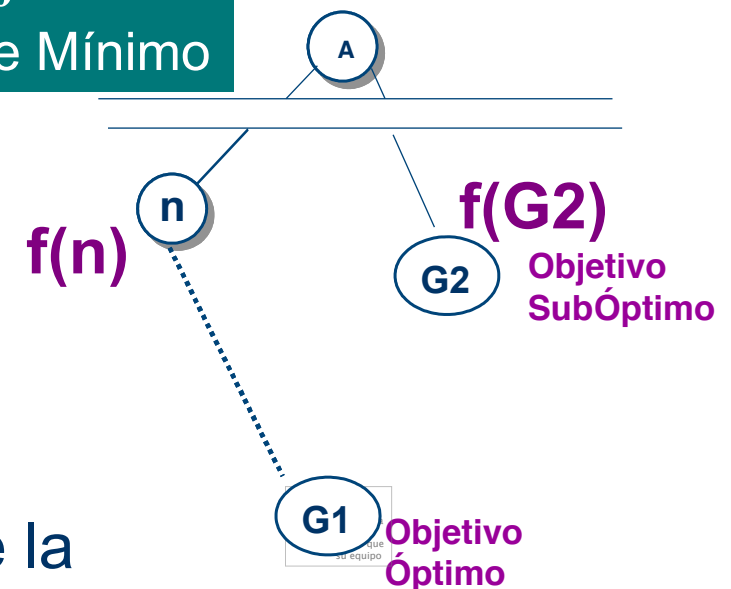
g(G2) MAYOR que C*

- n es un nodo en el camino de la solución óptima

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

h admisible

f(n) MENOR O IGUAL que C*

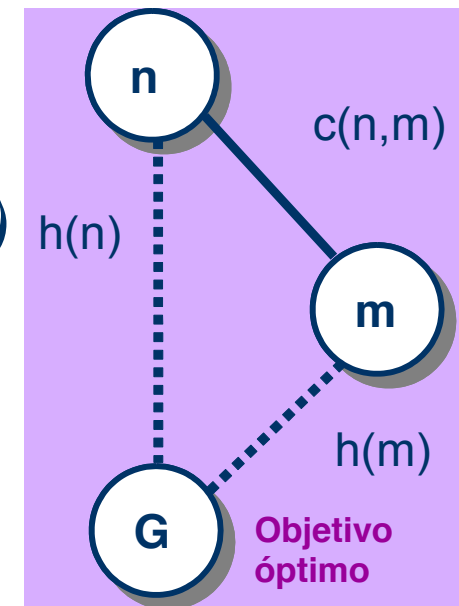


A* nunca seleccionará el nodo G2 porque
 $f(n) \leq C^* < f(G2)$

Consistencia o monotonía

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

$$h(n) \leq h(m) + c(n,m), \quad \forall (n,m)$$

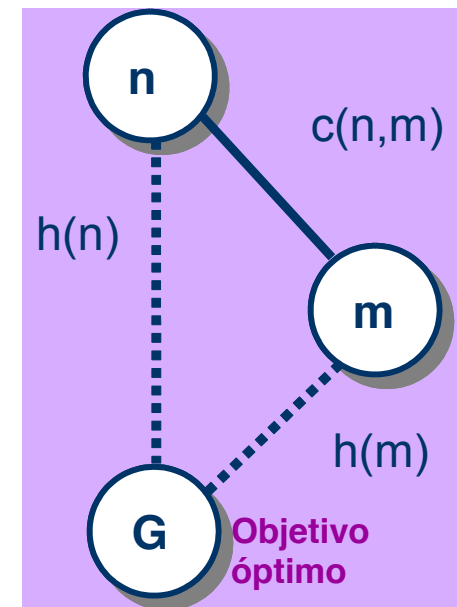


- **A* es óptimo si $h(n)$ es consistente** (con grafos de búsqueda) porque *se cumple esa desigualdad triangular*. **A* encuentra el camino óptimo.**
- Si $h(n)$ es *consistente* también es *admisibile*.

A* es Óptimo (Graf. Búsq.)

A*, usando grafos de búsqueda, es óptimo si:

- Se desecha el camino más caro.
- Se asegura que se sigue el camino óptimo a cualquier estado repetido, cumpliendo:
 - Función heurística consistente:
$$h(n) \leq h(m) + c(n,m), \quad \forall (n,m)$$
 - $f(n)$ es no decreciente



Si $h(n)$ es consistente entonces los valores de $f(n)$ a lo largo de cualquier camino no disminuyen.

Demostración A* es Óptimo (Graf. Búsq.)

Función heurística consistente:

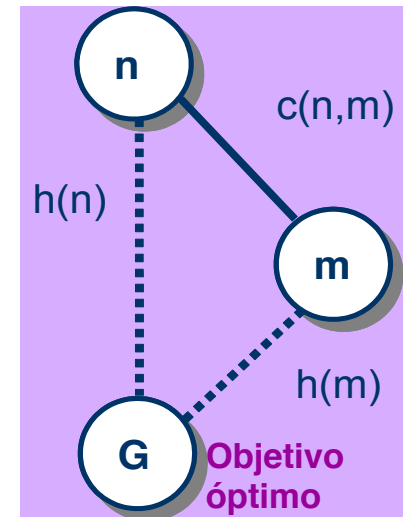
$$h(n) \leq h(m) + c(n, m), \quad \forall (n, m)$$

Si m es sucesor de n :

$$g(m) = g(n) + c(n, m)$$

Siguiendo la definición:

$$f(m) = g(m) + h(m) = g(n) + c(n, m) + h(m) \geq g(n) + h(n) = f(n)$$

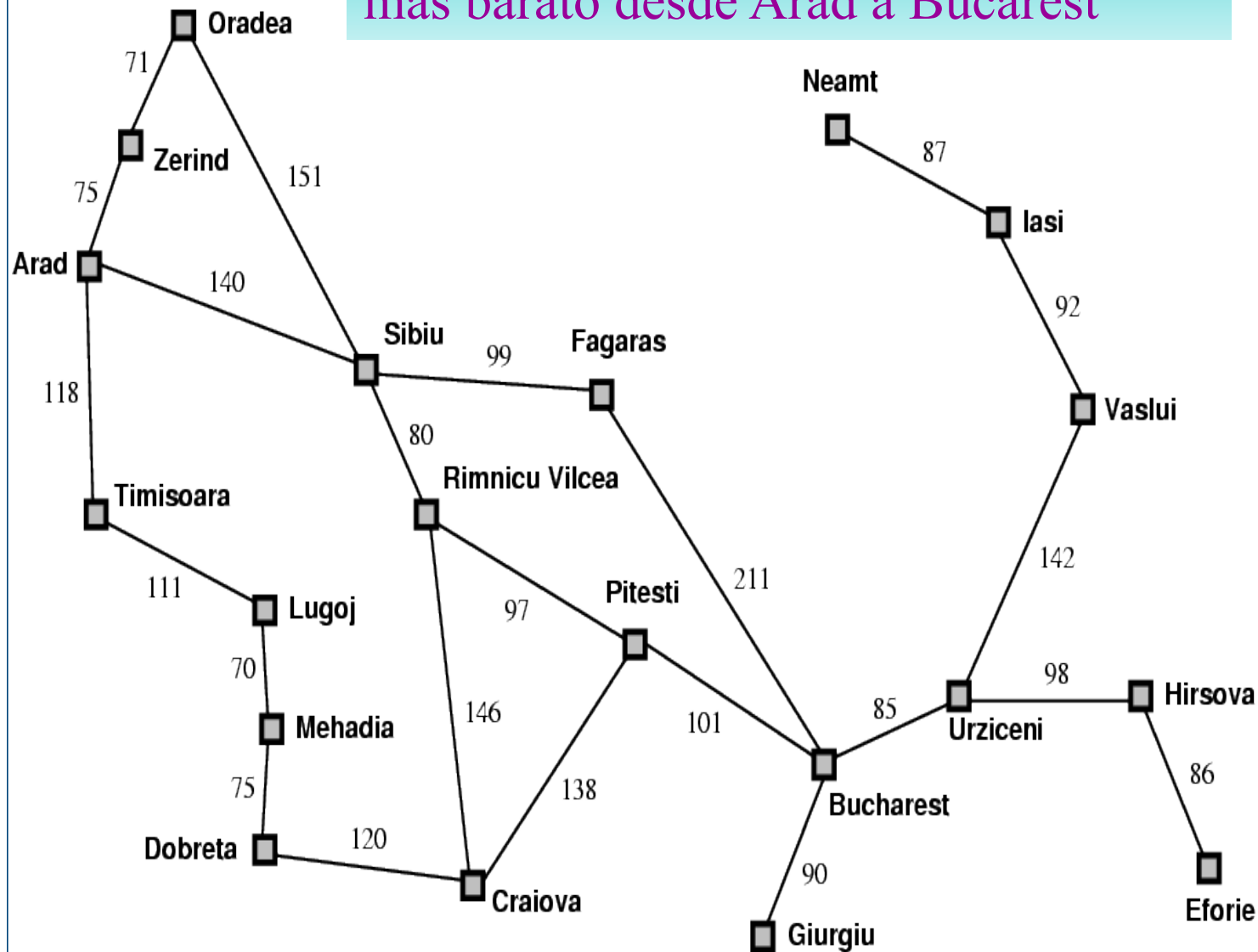


$f(n)$ MENOR O IGUAL que $f(m)$

Los nodos expandidos por A* tienen valores crecientes de $f(n)$. Por tanto, el primer nodo objetivo seleccionado para expandir es la solución óptima, ya que los demás nodos posteriores serán al menos tan costosos.

Si $h(n)$ es consistente A* encuentra el camino óptimo

Realizar una estimación del camino más barato desde Arad a Bucarest

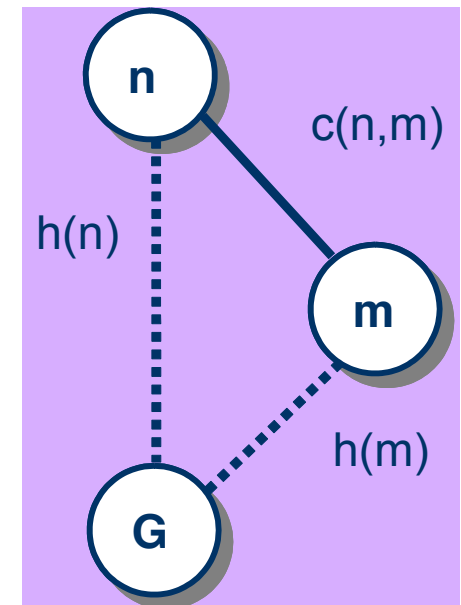
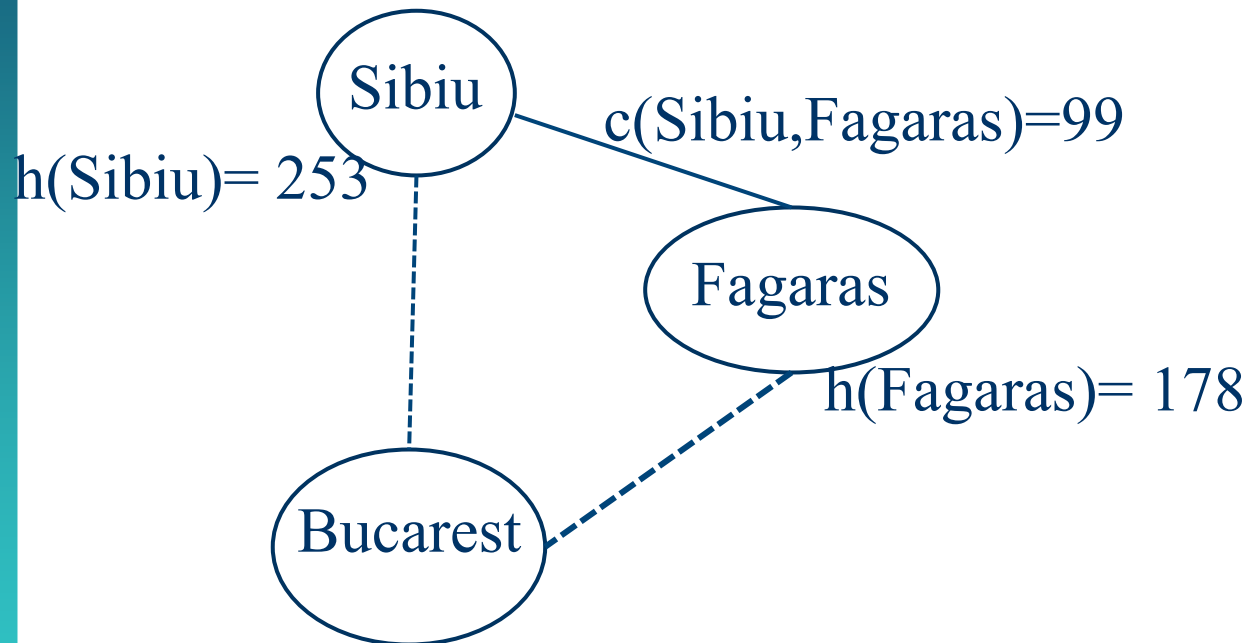


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

$h(n)$: distancia en línea recta de una ciudad a Bucarest

Consistencia en el ejemplo de las ciudades rumanas

- Heurística: distancia en línea recta



$$h(\text{Sibiu}) \leq h(\text{Fagaras}) + c(\text{Sibiu}, \text{Fagaras})$$

Heurísticas para el 8-puzzle

2	6	3
1		4
8	7	5

Estado Inicial



1	2	3
8		4
7	6	5

Estado Objetivo

Ambas son admisibles
¿Cuál es mejor?

- **$h1 = \text{n}^\circ \text{ de piezas mal colocadas}$**

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

- **$h2 = \text{suma de las distancias de Manhattan de las posiciones a sus objetivos.}$**

- La distancia de Manhattan es el n° 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.

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

Eficiencia en las Funciones Heurísticas

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

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



➡ Mejor heurística cuanto más cercano a 1 sea b^*

Funciones Heurísticas

- **Funciones Heurísticas Dominantes**

h_2 domina a h_1 (ambas admisibles) si

$$\forall n, h_2(n) \geq h_1(n) \quad \text{💬}$$

➡ h_2 nunca generará más nodos que h_1


Diseño de Funciones Heurísticas

- $h(n) = \max\{h_1(n), h_2(n), \dots, h_p(n)\}$, siendo h_1, h_2, \dots, h_p admisibles.
- Aprender $h(n)$ mediante la solución de muchos problemas (nodo, costo)
- Combinación de características :
$$h(n) = c_1x_1(n) + c_2x_2(n)$$

Diseño de Funciones Heurísticas

Relajación de precondiciones:

Ejemplo del 8-puzzle: Una ficha puede moverse del cuadrado A al cuadrado B si:

- A y B son adyacentes
- B está vacía 
- Problema relajado:
 - Si no son adyacentes cualquier ficha del tablero puede moverse al hueco

$h1 = n^{\circ}$ de piezas mal colocadas

- Una ficha se puede mover a una ficha adyacente que no esté vacía

$h2 = \sum \text{distancias de Manhattan de las posiciones a sus objetivos}$

Técnicas de Búsqueda Heurística

- **Búsqueda Primero el Mejor** (Best-First Search)
 - Búsqueda Voraz (Greedy search)
 - Algoritmo A*
- **Algoritmos Iterativos o de Búsqueda Local**
 - Búsqueda en escalada o Gradiente (Hill-climbing, Gradient descent)
 - Haz Local (Beam search)

Técnicas de Búsqueda Heurística

1. BÚSQUEDA PRIMERO EL MEJOR (Best-First Search)

- Búsqueda Voraz o Avara (Greedy search)
- Algoritmo A*
- Mejoras al Algoritmo A*

2. FUNCIONES HEURÍSTICAS

- Propiedades

3. BÚSQUEDA LOCAL

- Búsqueda en escalada o Gradiente (Hill-climbing, Gradient descent)
- Haz Local (Beam search)

Algoritmos de 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 sólo a los nodos vecinos del nodo actual
 - No son sistemáticos en la búsqueda
- Utilizan poca memoria
- Pueden encontrar soluciones razonables en espacios de estados grandes o infinitos
- Pueden quedar atrapados en máximos/mínimos locales

Búsqueda en Escalada

- Bucle que continuamente se mueve en la dirección de un valor:
 - creciente (si se trata de maximizar una función objetivo)
 - decreciente (si se trata de minimizar la función de coste)
- Se generan los sucesores de un estado n , y se devuelve m de la expansión, por ser el que tiene mejor valor de la función de evaluación:

$$f(m) > f(n) \text{ (creciente)}$$

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

Búsqueda en Escalada

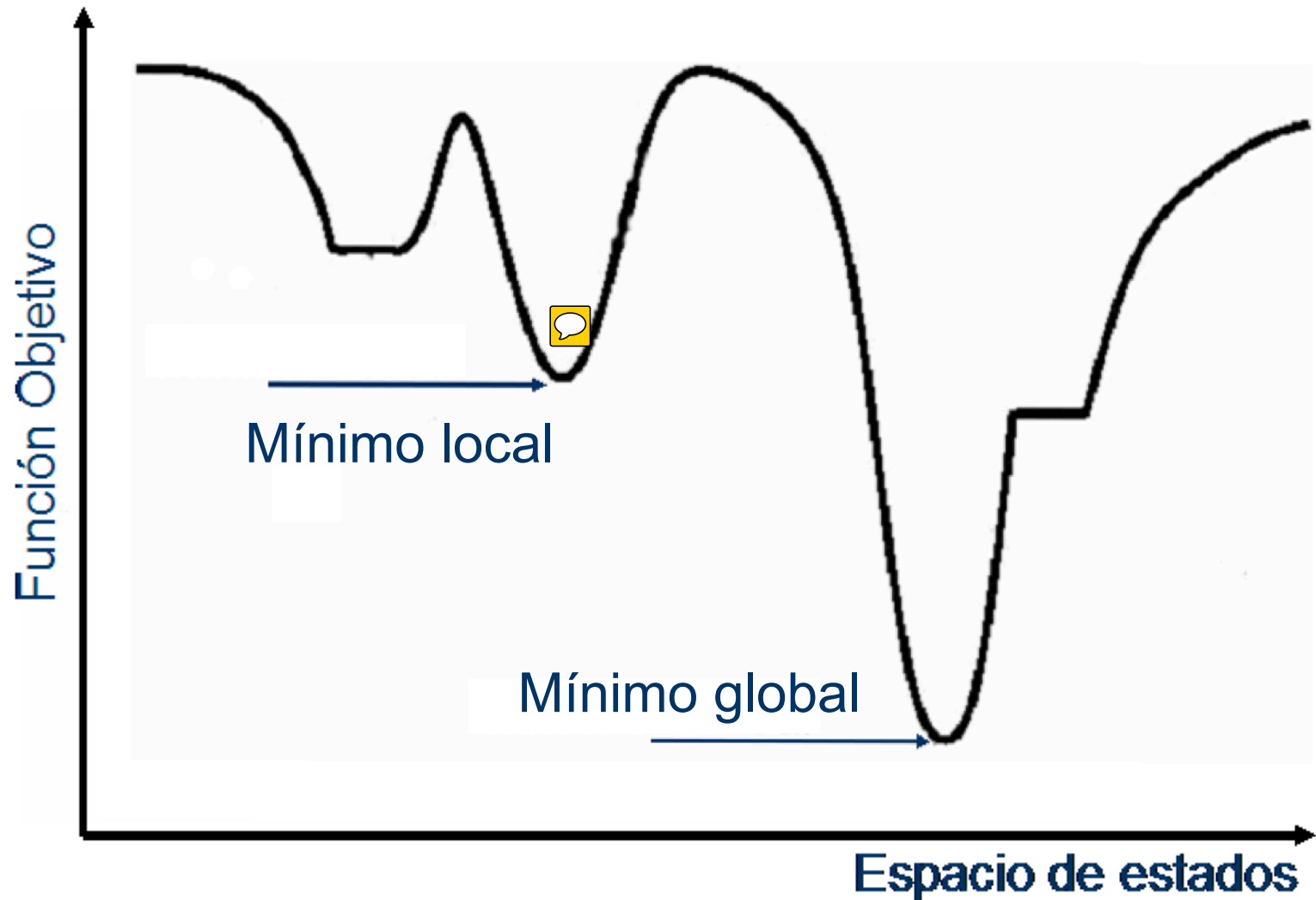
- Sigue el recorrido a través de los nodos en los que el valor de dicha función sea máximo (cuesta arriba) o mínimo (depende del planteamiento).
- No mantiene un árbol de búsqueda, tan sólo una estructura con el estado y el valor de la función objetivo.
- La Búsqueda en Escalada sólo mira a los vecinos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

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 mayor  
si f(vecino) < f(actual) (DECRECIANTE)  
  entonces ABIERTOS:= vecino
```

Problemas



Rendimiento de Búsq. en Escalada

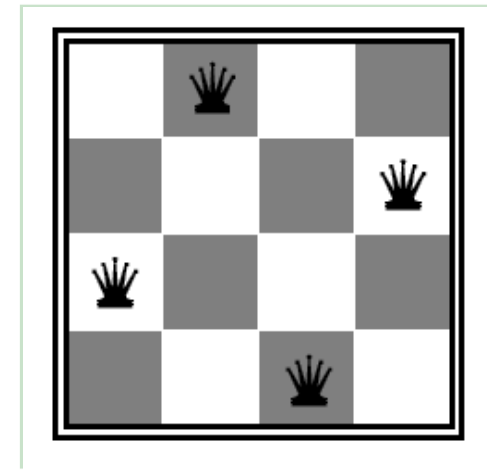
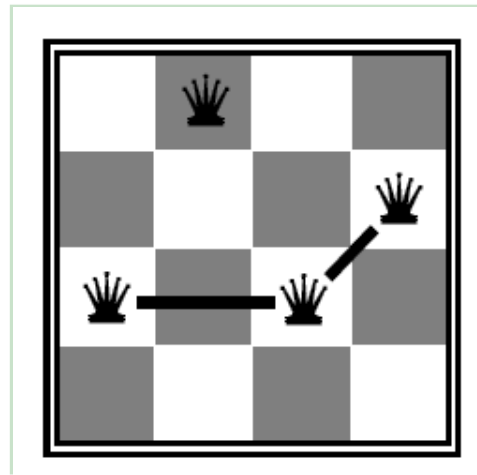
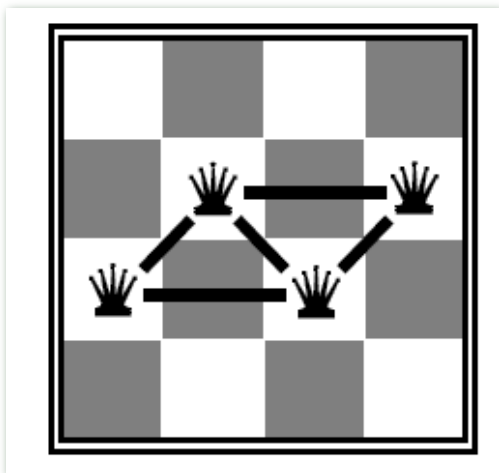
- **No es completa**
 - **No es óptima**
 - Pero puede encontrar soluciones aceptables
 - **Complejidad en tiempo:**
nº de nodos generados $O(b \cdot m)$
 - **Complejidad en espacio:**
Almacena un estado $O(1)$
- (**b**: factor de ramificación, **m**: profundidad máxima y **d**: profundidad de la solución óptima)

Búsqueda por Haz Local

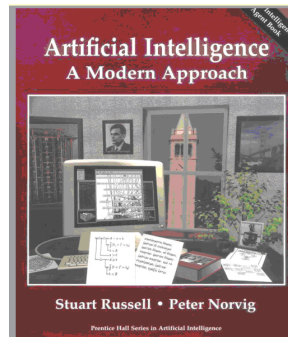
- Guarda la pista de k estados.
- Comienza con estados generados aleatoriamente
- En cada paso, se generan todos los sucesores de los k estados.
- Si alguno es un objetivo, finaliza.
- Si no, se seleccionan los k mejores sucesores de la lista completa y se repite el proceso

El problema de las N-Reinas

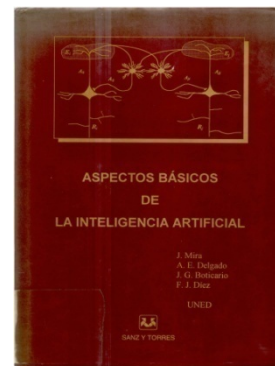
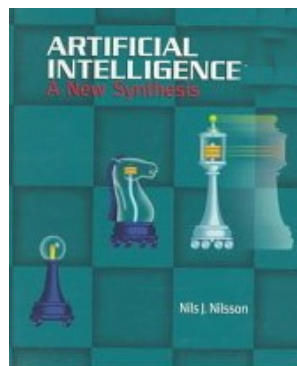
Colocar n reinas sobre un tablero de $n \times n$, sin que queden dos reinas en la misma columna, fila, o diagonal



Referencias Bibliográficas



- **Inteligencia Artificial: Un Enfoque Moderno.** S. Russell y P. Norvig, 2005
- **Problemas Resueltos de IA Aplicada. Búsqueda y Representación.** Fernández et al. (2003)



- **Aspectos básicos de la Inteligencia Artificial.** Mira et al. , 2003