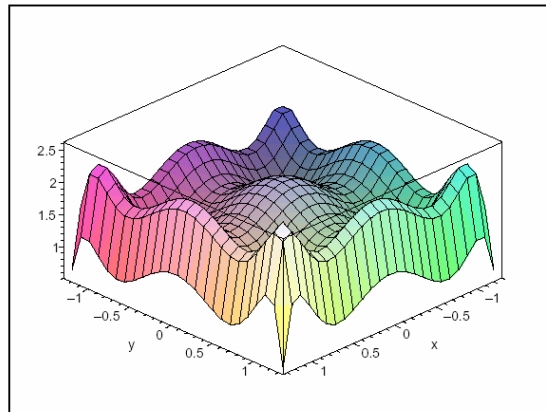


# Búsqueda con información, informada o heurística



# Heurística

- Del griego *heuriskein* (encontrar, descubrir).
  - » Arquímedes → ¡EUREKA!
  - » Uso en IA
    - 1957, (G. Polya): Estudio de métodos para descubrir formas de resolución de problemas
    - 1963, (Newell): Proceso que puede resolver un problema pero sin garantías de que lo haga
    - El 1er. Laboratorio de Sistemas Expertos (en Stanford) se denominó HPP: *Heuristic Programming Project*
    - Actualmente:
      - Cualquier técnica que mejore la ejecución en el caso promedio durante las tareas de resolución de problemas, pero que no mejore necesariamente el peor caso.

# Estrategias de búsqueda informada

- Estrategias que usan la información de definición del problema y el coste del estado actual al objetivo (información específica del problema)
- Estrategias:
  - » El primero mejor (*Best first Search*)
  - » Búsqueda Avara
  - » A\*
  - » IDA\*
  - » Mejora iterativa
    - *Hill climbing*
    - *Simulated Annealing*

# Búsqueda el primero mejor, I

- Búsqueda el primero mejor
  - » Se incorpora una función de evaluación (*eval-fn*) que mide lo deseable de expandir un nodo.
    - Se expande el nodo con  $f(n)$  menor
    - Best-first se puede implementar como una *cola de prioridad*, estructura de datos que mantiene la frontera en orden ascendente de los valores de  $f$
    - Existe una variedad importante de algoritmos el-primero-mejor con diferentes funciones de evaluación. Una componente esencial de los mismos es la **función heurística  $h(n)$** .
      - $h(n)$  = valor estimado del camino de coste mínimo desde el nodo  $n$  al nodo objetivo
      - Todas las funciones heurísticas deben cumplir:
        - »  $h(n) \geq 0$ , para todo nodo  $n$
        - »  $h(n) = 0$ , si  $n$  es un nodo objetivo

# Búsqueda el primero mejor, II

A partir del algoritmo de búsqueda general, introducimos conocimiento específico del problema al insertar los nodos sucesores en la cola mediante una *función de evaluación*.

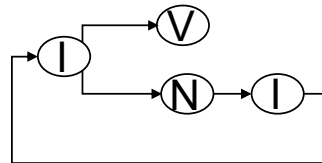
Función evaluación: medida de lo “deseable” de expandir un nodo.

```
function BEST-FIRST-SEARCH(problem, EVAL-FN) returns a solution sequence
inputs: problem, a problem
        Eval-Fn, an evaluation function
        Queueing-Fn ← a function that orders nodes by EVAL-FN
return GENERAL-SEARCH(problem, Queueing-Fn)
```

Se expande primero el nodo no expandido más “deseable”

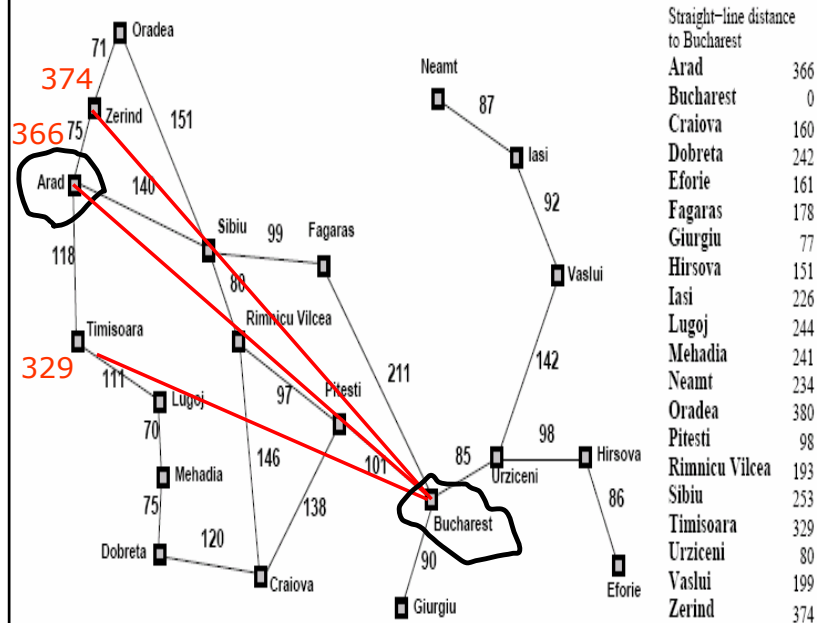
# Búsqueda avara, I

- Búsqueda el primero mejor donde
  - »  $\text{eval-fn}(\text{nodo}) = h(\text{nodo})$
- Suele encontrar soluciones rápido
  - » No suelen ser óptimas
  - » No siempre se encuentran (estados repetidos  $\rightarrow$  ciclos)
    - Ej. de situación anómala: Ir de *Iasi* a *Fagaras*. Si no eliminamos repeticiones se entra en un ciclo.



- » Ejemplo: Mapa de carreteras
  - Objetivo: Viajar desde Arad hasta Bucarest.
  - Heurística  $h$ : distancias medidas en línea recta (sobre el mapa) entre Arad y Bucarest
  - Solución obtenida por búsqueda avara:
    - Nodos expandidos Encuentra el camino
      - » “Arad, Sibiu, Fagaras, Bucharest”,
    - No es óptima: Es más corto
      - » “Arad, Sibiu, Rimnicu, Pitesti, Bucharest”

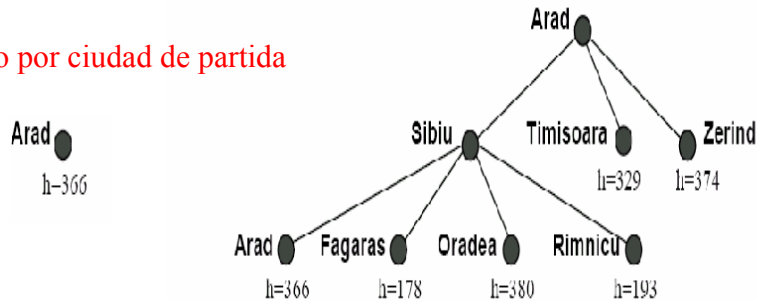
## Romania with step costs in km



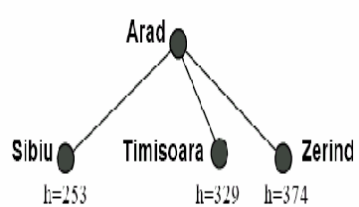
# Búsqueda avara: ejemplo del mapa de carreteras

2. Expandir: Sibiu (h menor, 253)

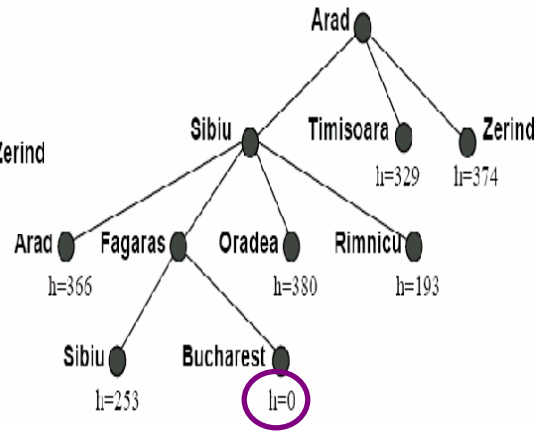
Inicio por ciudad de partida



1. Expandir: Arad



3. Expandir: Fagaras (h menor, 178)



4. Se llega a Bucharest, solución encontrada

## Búsqueda avara, II

- En resumen:
  - » No es óptimo ni completo.
  - » En el peor caso:
    - Complejidad temporal:  $O(b^m)$
    - Complejidad espacial:  $O(b^m)$ 
      - Se almacenan todos los nodos en memoria
      - $m$  = máxima profundidad del árbol de búsqueda

# Algoritmo A\*, I

- Algoritmo A\*, combinación de:
  - » Búsqueda avara:
    - Reduce coste de búsqueda, pero no es óptima ni completa.
  - » Búsqueda de coste uniforme:
    - Es completa y óptima pero ineficiente.
- Se define la función  $f(n)$ :
  - »  $f(n)=g(n)+h(n)$
  - »  $f(n)$ =coste estimado de la solución de menor coste que pasa por  $n$
- Algoritmo:

```
function A*-SEARCH (problem)  
returns a solution or failure  
    return BEST-FIRST-SEARCH(problem,  $g+h$ )
```

# Algoritmo A\*, II

- Heurística admisible:

- » DEF: Una función heurística  $h$  es admisible si

$$h(n) \leq h^*(n), \forall n$$

- » en donde  $h^*(n)$ ="mínima distancia desde  $n$  hasta el objetivo"

- » Las heurísticas admisibles son optimistas en el sentido de que el coste de alcanzar el objetivo  $h(n)$ , es menor de lo que lo es actualmente  $h^*(n)$ .

- Ejemplo:

- » En el mapa de carreteras,  $h$  es admisible.

- » Solución obtenida por A\*:

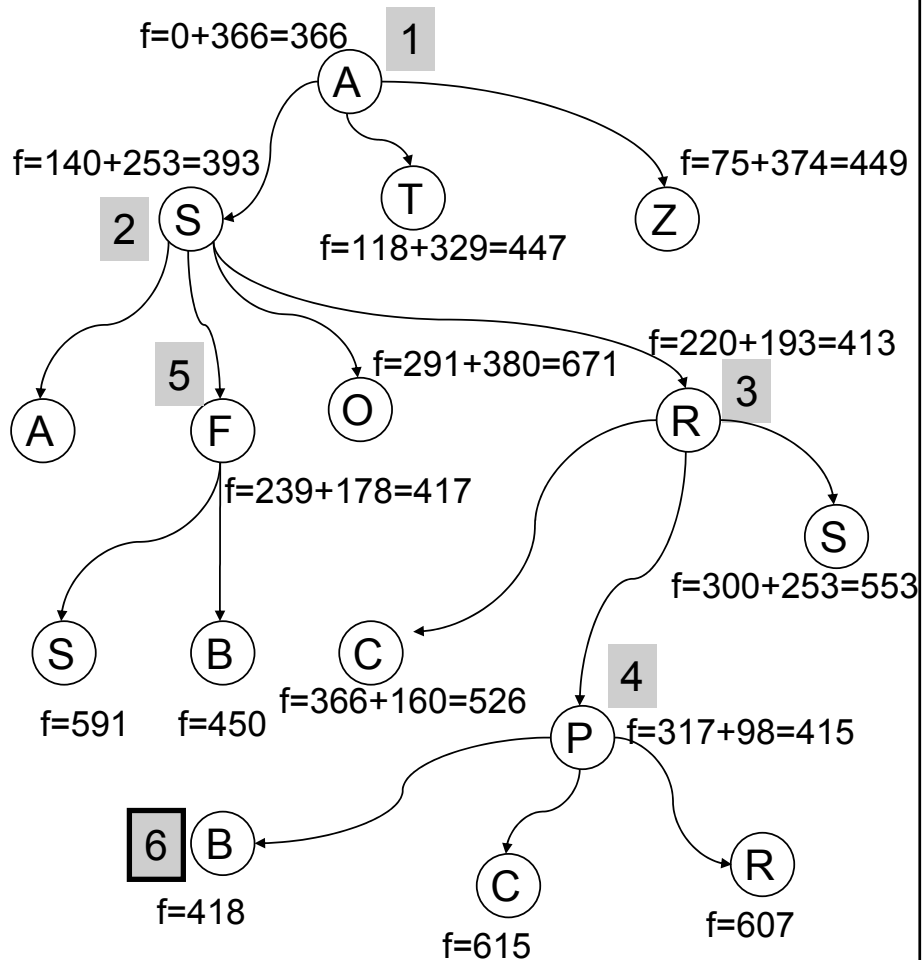
- Orden de expansión: "A, S, R, P, F, B"

- Encuentra la solución: "A, S, R, P, B"

- Aplicación algoritmo (ver siguiente página)

- Es la mejor solución (A\* es óptimo)

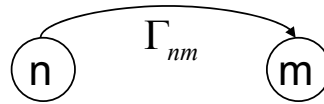
# Algoritmo A\*, III



# Algoritmo A\*, IV

- Una heurística es monótona cuando:

$$\forall \Gamma_{nm}, h(n) - h(m) \leq \text{coste}(\Gamma_{nm})$$



- Si  $h$  es monótona  $\Rightarrow h$  admisible.

» **Dems:**

» Sea  $n$  un nodo, y sea  $\Gamma$  un camino desde  $n$  hasta el objetivo:

$$\Gamma = \Gamma_{n_0 n_1 \dots n_k}$$

donde  $n_0 = n$  y  $n_k$  es un nodo objetivo.

$$\begin{aligned} h(n) - h(n_k) &= h(n_0) - h(n_k) = \\ &= h(n_0) - h(n_1) + h(n_1) - \dots - h(n_{k-1}) + h(n_{k-1}) - h(n_k) \leq \\ &\leq \text{coste}(\Gamma_{n_0 n_1}) + \dots + \text{coste}(\Gamma_{n_{k-1} n_k}) = \text{coste}(\Gamma_{n_0 n_k}) = \text{coste}(\Gamma) \end{aligned}$$

Por tanto

h monótona

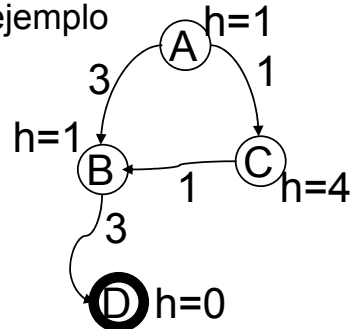
$$h(n_0) - 0 = h(n) \leq \text{coste}(\Gamma), \forall \Gamma \Rightarrow$$

$$\Rightarrow h(n) \leq h^*(n), \forall n$$

# Algoritmo A\*, V

» Teorema:  $h$  admisible  $\not\Rightarrow$  monótona

– Dems: Contraejemplo



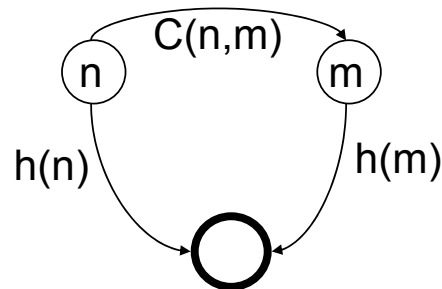
» Teorema:  $h$  heurística monótona  $\Leftrightarrow f$  creciente

– En el problema del mapa,

- $h$  es monótona ( $d$ : distancia en línea recta; “C”: coste del arco)

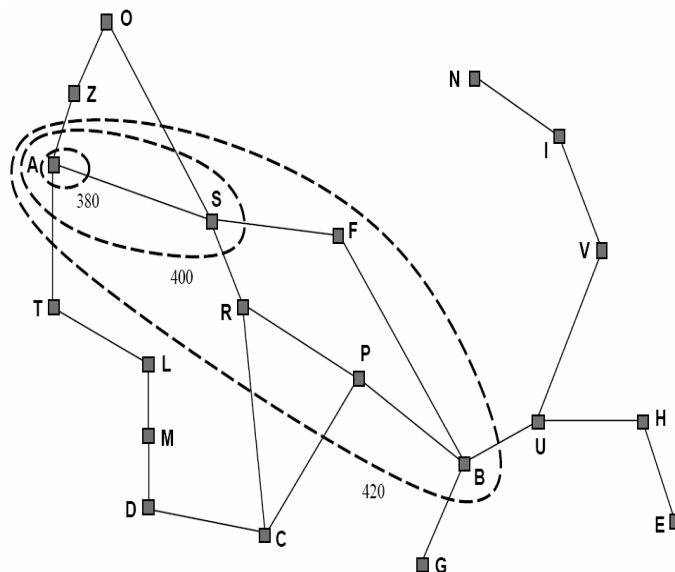
$$h(A) - h(B) \leq d(A, B) \leq C(A, B)$$

- $f$  es creciente (ver gráfico del ejemplo)



# Propiedades de A\*, I

- Teorema: A\* es óptimo y completo si  $h$  es admisible
  - » Válido en *grafos localmente finitos*
    - con factores de ramificación finitos
    - donde para todo operador:  $C(\Gamma) \geq \delta > 0, \forall \Gamma$
  - » Conjuntos de nivel (caso de heurísticas monótonas):
    - Si búsqueda uniforme ( $h(n)=0$ ): bandas circulares
    - si la heurística es más exacta ( $h \rightarrow h^*$ ), los conjuntos de nivel se dirigen directamente al objetivo.



# Propiedades de A\*, II

» Dems: A\* es óptimo

» Hipótesis

– (1)  $h$  es admisible

– (2)  $G$  es óptimo con coste de camino  $f^*$

– (3)  $G'$  es objetivo subóptimo  $g(G') > f^*$

– **Dems:**

Como  $G'$  es subóptimo

$$f(G') = g(G') + h(G') = g(G') > f^*$$

Supongamos que el nodo  $n$  está en el camino al óptimo

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

Por tanto

$$f(n) \leq f^* < f(G') = g(G') + h(G') = g(G')$$

que es una contradicción con (3). cqd.

# Propiedades de $A^*$ , III

» Dems:  $A^*$  es completo

» Hipótesis:

- (1) heurísticas monótonas,
- (2) factor de ramificación  $b$ ,  $b < \infty$
- (3) coste de operadores,  $C(\Gamma) \geq \delta > 0, \forall \Gamma$
- (4)  $f^*$  es finito
- **Dems:** En algún momento se llegará a que  $f = \text{“coste de algún estado objetivo”}$ , salvo que existan infinitos nodos con  $f(n) < f^*$ , lo cual sucedería si:

- Un nodo tuviera  $b = \infty$  (contradice (2))
- Hubiera un camino de coste finito pero con infinitos nodos. Esto significaría que, por (1), (3) y (4)

$$\exists n / f(n) > f^*$$

Por tanto, el algoritmo debe acabar. Y si acaba, es que encuentra una solución. cqd.

## Propiedades de A\*, IV

- Proposición: Si  $h$  es monótona, y  $A^*$  ha expandido un nodo  $n$ , entonces  $g(n)=g^*(n)$ 
  - » Es consecuencia directa de que:
    - Un subgrafo de una heurística monótona da lugar a una heurística monótona (que es, por tanto, admisible), considerando la nueva heurística  $h'=h-g(n)$
    - $h$  admisible  $\rightarrow A^*$  completo y óptimo
- $A^*$  es óptimamente eficiente
  - » Ningún otro algoritmo óptimo expandirá menos nodos que  $A^*$  (salvo *muerte súbita* entre nodos  $n$  con  $f(n)=f^*$ )
    - Si algún algoritmo expande menos nodos corre el riesgo de perder la solución óptima
  - » Si un algoritmo no expande todos los nodos entre el origen y el contorno óptimo, corre el riesgo de perder la solución óptima.
    - Dems: ver Dechter – Pearl (1985)

# Propiedades de $A^*$ , $V$

» Complejidad (temporal y espacial):

$$O(b^{\tilde{d}}), \tilde{d} = \frac{f^*}{\text{minimo - valor - costes}}$$

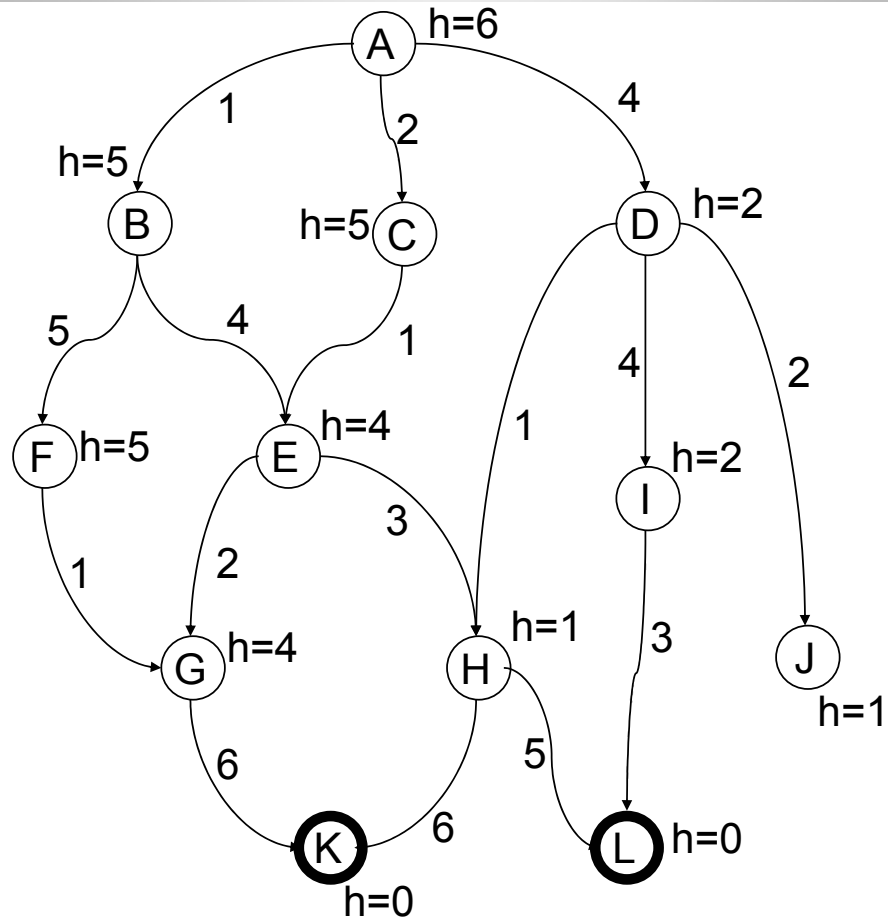
- Se puede demostrar que la complejidad del algoritmo sigue siendo exponencial salvo que el error en la función heurística no crezca más rápido que el logaritmo del coste del camino óptimo, es decir:

$$|h(n) - h^*(n)| \leq O(\log h^*(n)), \forall n$$

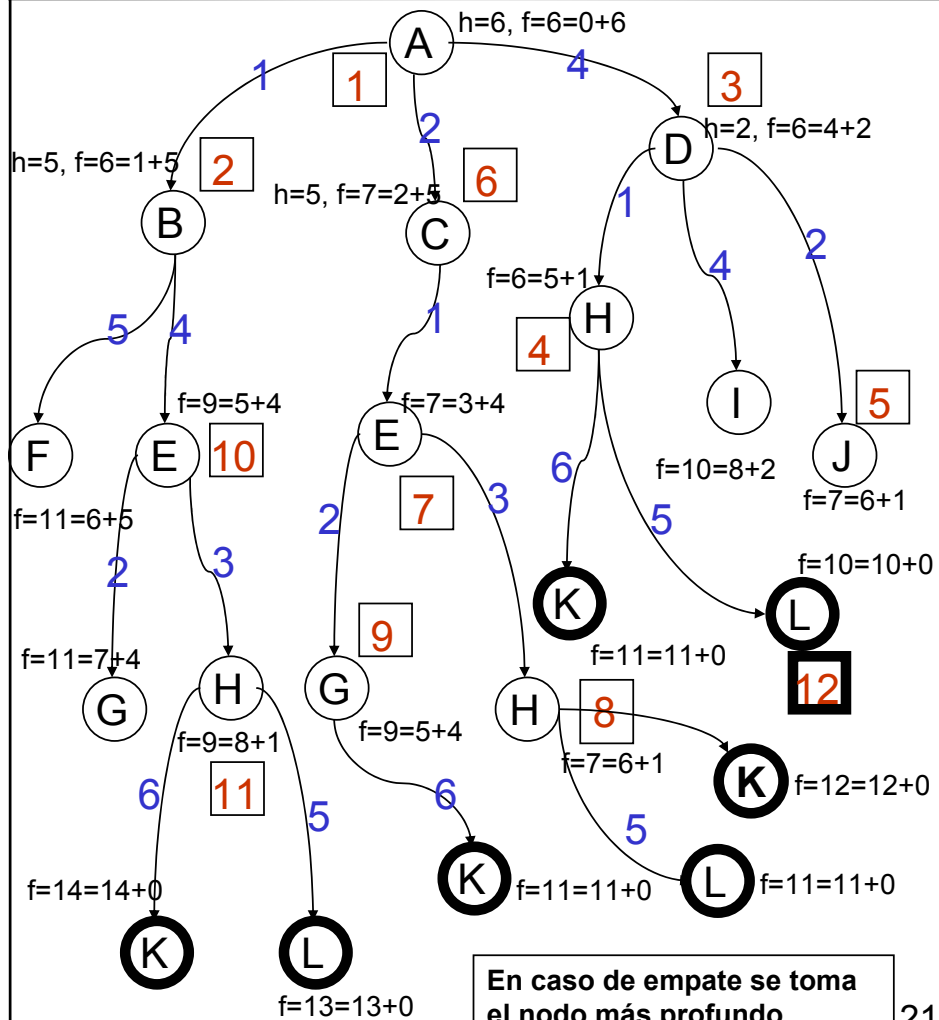
Siendo  $h^*(n)$  el coste óptimo de alcanzar el objetivo desde  $n$ .

- En casi todas las heurísticas, el error es al menos proporcional al coste del camino y, por tanto, se tiene complejidad exponencial.
- De todos modos, el uso de heurísticas produce enormes mejoras con respecto a la búsqueda no informada.
- La complejidad espacial suele ser un mayor problema que la temporal, al requerir mantener en memoria todos los nodos generados.

# Un ejemplo de A\*, I

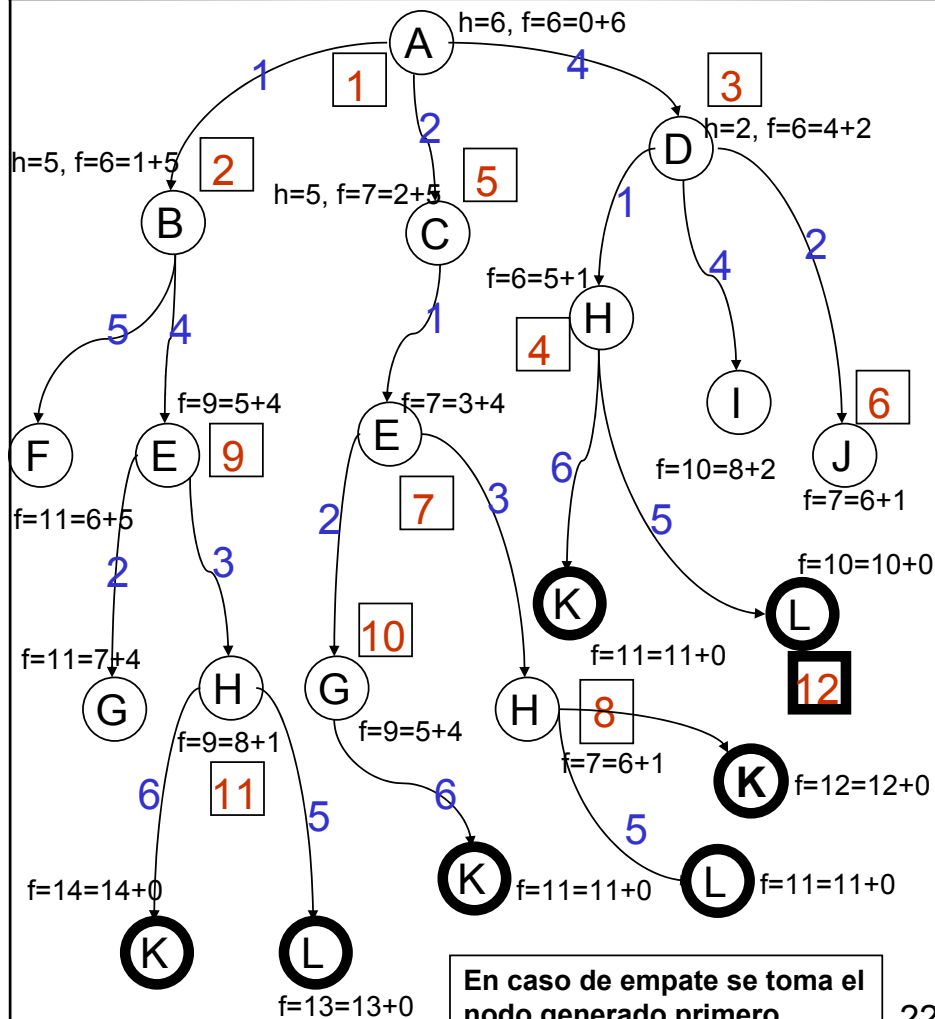


# Un ejemplo de A\*, Ila



En caso de empate se toma el nodo más profundo

# Un ejemplo de A\*, IIb



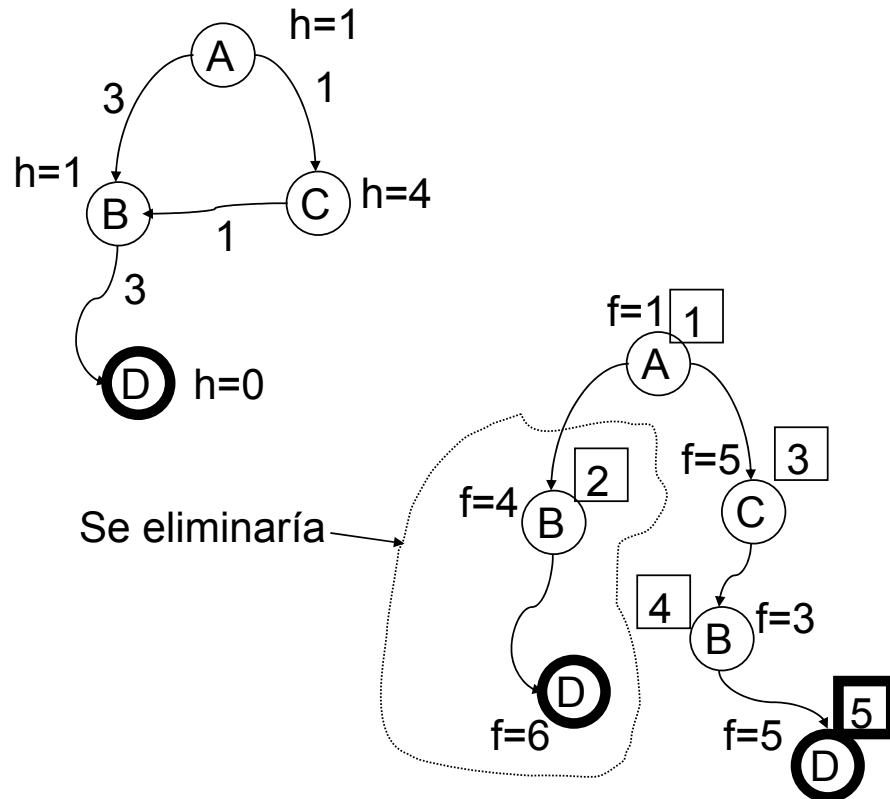
En caso de empate se toma el nodo generado primero

## Un ejemplo de A\*, III

- Con eliminación de estados repetidos:
  - » eliminar expansiones: (B→E), (E→H), (G→K)
- Al ser h monótona:
  - » A\* obtiene la mejor solución al eliminar estados repetidos, ya que:
    - Si h es monótona, entonces si un nodo ha sido expandido  $\rightarrow g(n)=g^*(n)$   
entonces, bastaría con:
      - 1) Si está repetido en los nodos ya expandidos, eliminar directamente el “nodo nuevo”
      - 2) Si está repetido en las hojas, quedarse con el mejor entre el nodo “viejo” y el “nuevo”

# Otros ejemplos de A\*, I

- Eliminando estados repetidos, caso de  $h$  admisible no monótona, podemos eliminar un nodo peor ya expandido:



# Exactitud de heurísticas, I

- Factor efectivo de ramificación  $b^*$ :
  - » Medida de la calidad de una heurística
  - »  $N$  = número de nodos expandidos por  $A^*$  (incluido el nodo de la mejor solución)
  - »  $d$  = profundidad de la solución obtenida por  $A^*$
  - »  $b^*$  = factor de ramificación de un árbol de profundidad  $d$  y  $N + 1$  nodos.

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d = \frac{(b^*)^{d+1} - 1}{b^* - 1}$$

- » Resolución no algebraica, sino con métodos de cálculo numérico.
- » Ejemplo:  $d=5, N=52 \rightarrow b^* = 1.92$
- » Normalmente, para una heurística  $h$ ,  $b^* \cong$  constante en varias instancias del problema.
- » La bondad de una heurística se mide por la aproximación  $b^* \rightarrow 1$
- » Si  $h \rightarrow h^*$ , entonces  $b^* \rightarrow 1$
- » Si  $h \rightarrow 0$  (coste uniforme), entonces  $b^* \rightarrow b$

## Exactitud de heurísticas, II

- Dominación

$$h_2 \succ h_1 \Rightarrow \forall n, h_2(n) \geq h_1(n)$$

- » Si una heurística  $h_2$  domina a otra  $h_1$  (supuestas ambas monótonas), entonces  $h_1$  expande, al menos, los mismos nodos que  $h_2$ .

- » Idea intuitiva:

- Si  $f(n) < f^*$ , entonces  $n$  se expande. Pero esto es equivalente a  $h(n) < f^* - g(n)$ . Por tanto, si ese nodo es expandido por  $h_2$ , también lo es por  $h_1$

## Ejemplo de dominancia I, (*8-puzzle*)

- Análisis del problema:
  - » Una solución típica realiza unas 22 acciones
  - » Factor de ramificación:  $b \approx 3$ 
    - Hueco en centro → 4 movimientos
    - Hueco en lado → 3 movimientos
    - Hueco en esquina → 2 movimientos
  - » Complejidad:
    - La búsqueda exhaustiva hasta  $d = 22$  requiere visitar
$$3^{22} = 3.1 * 10^{10} \text{ estados}$$
    - Si se consideran estados repetidos, el número de estados distintos a visitar es  $9!/2=181.440$  (reducción de factor 170.000)

## Ejemplo de dominancia II, (8-puzzle)

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

$h_1(\text{start}) = 7$

(Errores posición: número fichas mal colocadas)

$h_2(\text{start}) = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$

(suma de distancias de *Manhattan*)

$h_1$  y  $h_2$  monótonas

$$\forall n, h_2(n) \geq h_1(n) \Rightarrow h_2 \text{ domina a } h_1$$

Observaciones:

1.  $h_1(n)$  expande al menos los mismos nodos que  $h_2(n)$ , si un nodo es expandido por  $h_2(n)$  también es expandido por  $h_1(n)$ .
2. La dominación se traduce en eficiencia: una heurística dominante expande menos nodos.
3. Lo mejor es usar una heurística dominante siempre y cuando sea admisible.

# Creación de funciones heurísticas

- Método de relajación
  - » Un problema con menos restricciones sobre las acciones se denomina *relajado*
  - » El coste de la solución óptima de un problema relajado es una heurística admisible para el problema original (la heurística derivada es consistente, verifica la desigualdad triangular)
  - » Ejemplo:
    - problema original (*8-puzzle*):
      - Si A es adyacente (vertical u horizontal) a B y B es blanco, entonces mueve ficha desde A hasta B
    - Problemas relajados:
      - 1) Si A es adyacente a B, entonces mueve ficha desde A hasta B
      - 2) Si B es blanco, entonces mueve ficha desde A hasta B
      - 3) mueve ficha desde A hasta B
    - » Heurísticas:
      - $h^*$  para problema 1) = heurística  $h_2$  (*Manhattan*)
      - $h^*$  para problema 3) = heurística  $h_1$  (errores posición)
      - $h^*$  para problema 2) = heurística de Gaschnig:  
Número mínimo de movimientos necesarios para alcanzar el objetivo. Consideramos un movimiento coger una ficha de una posición y depositarla en el hueco vacío.
- Método de la heurística compuesta:
  - » Si  $h_1, h_2, \dots, h_n$  son admisibles entonces  $h(n) = \max\{h_1, h_2, \dots, h_n\}$  también es admisible

# Algoritmo IDA\*, I

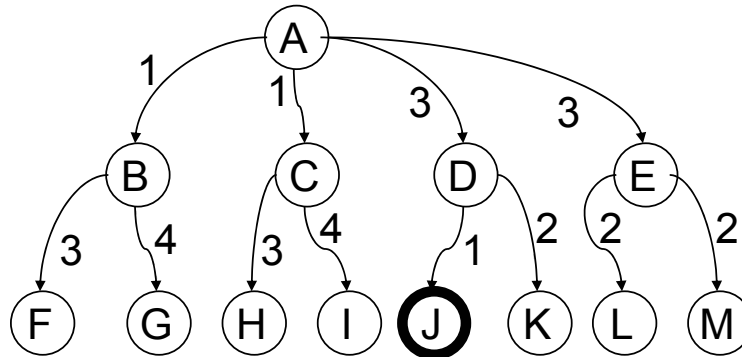
- Iterative-Deepening A\*- search
  - » Adapta a A\* la idea de búsqueda en profundidad iterativa
  - » El criterio de corte es el coste  $f = g + h$  (no la profundidad)
  - » Útil para problemas de costes unitarios
  - »  $h$  monótona
- Algoritmo
  - »  $s$  = nodo inicial
  - » Se definen:  
$$C_0 = f(s), \quad \text{si } k = 0$$
$$C_k = \min_n \{f(n) / f(n) > C_{k-1}\}, \quad \text{si } k \geq 1$$
  - » El algoritmo realiza búsqueda en profundidad para los valores  $L=0,1,2,\dots$  en los conjuntos:

$$K_L = \{n / f(n) \leq C_L\}$$

# Ejemplos IDA\*, I

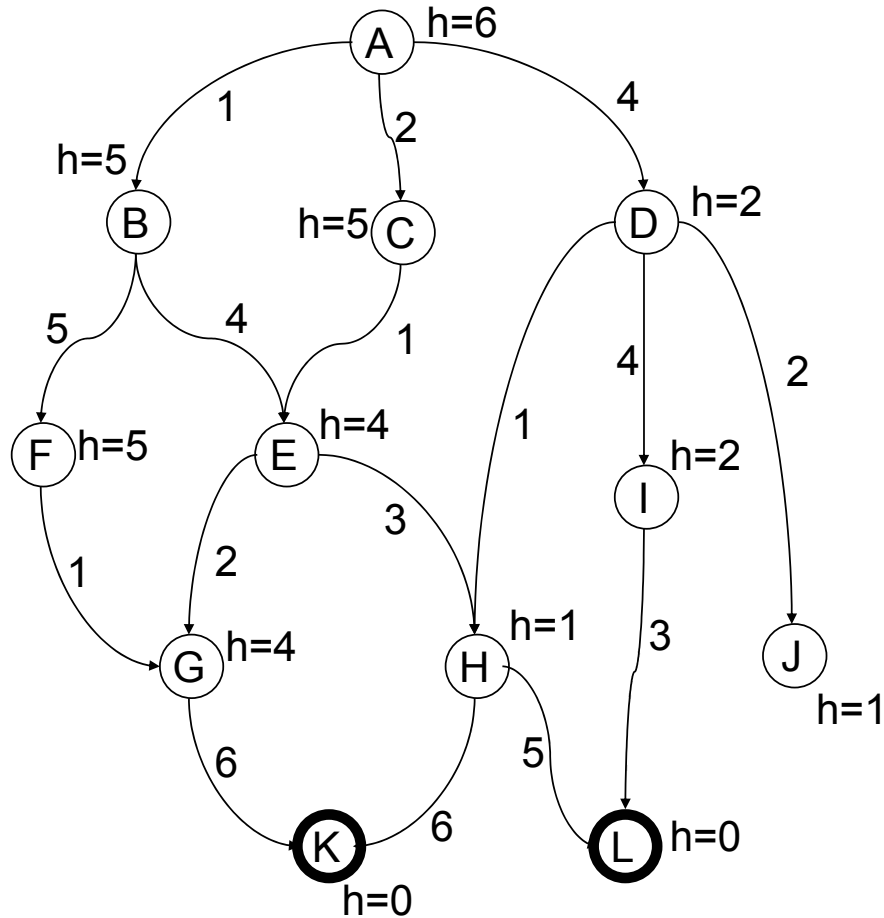
- Ejemplo simple ( $h = 0$ )
  - » Se producen 4 iteraciones

$$f \leq 0; f \leq 1; f \leq 3; f \leq 4$$

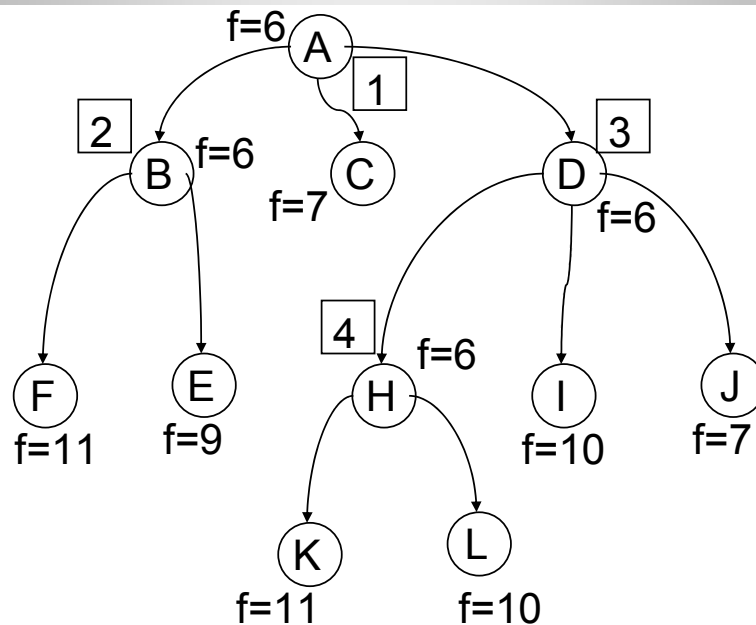


# Ejemplos IDA\*, II

- Ejemplo  $h$  no nula

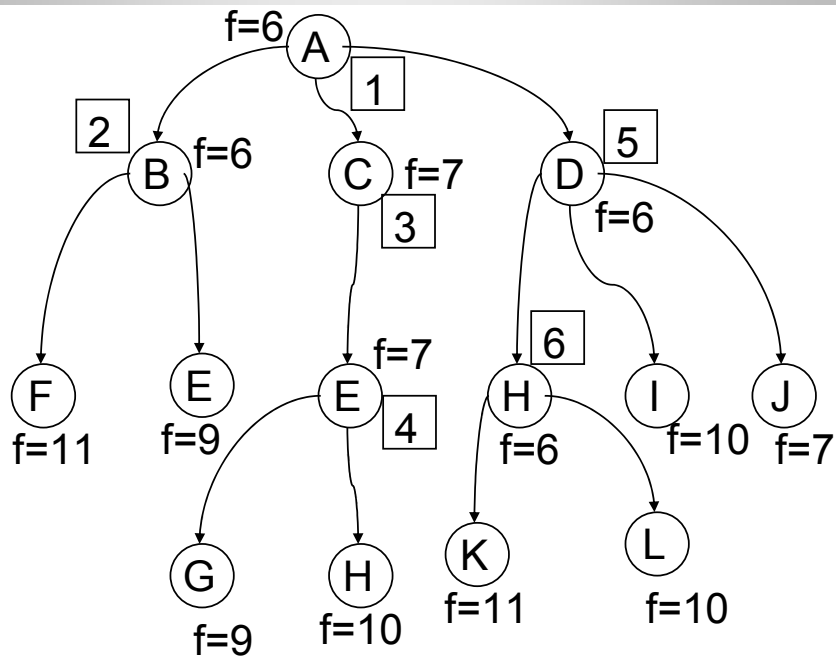


# Algoritmo IDA\*, III-a



Ejemplo de aplicación de IDA\*, iteración  $f \leq 6$

# Algoritmo IDA\*, III-b

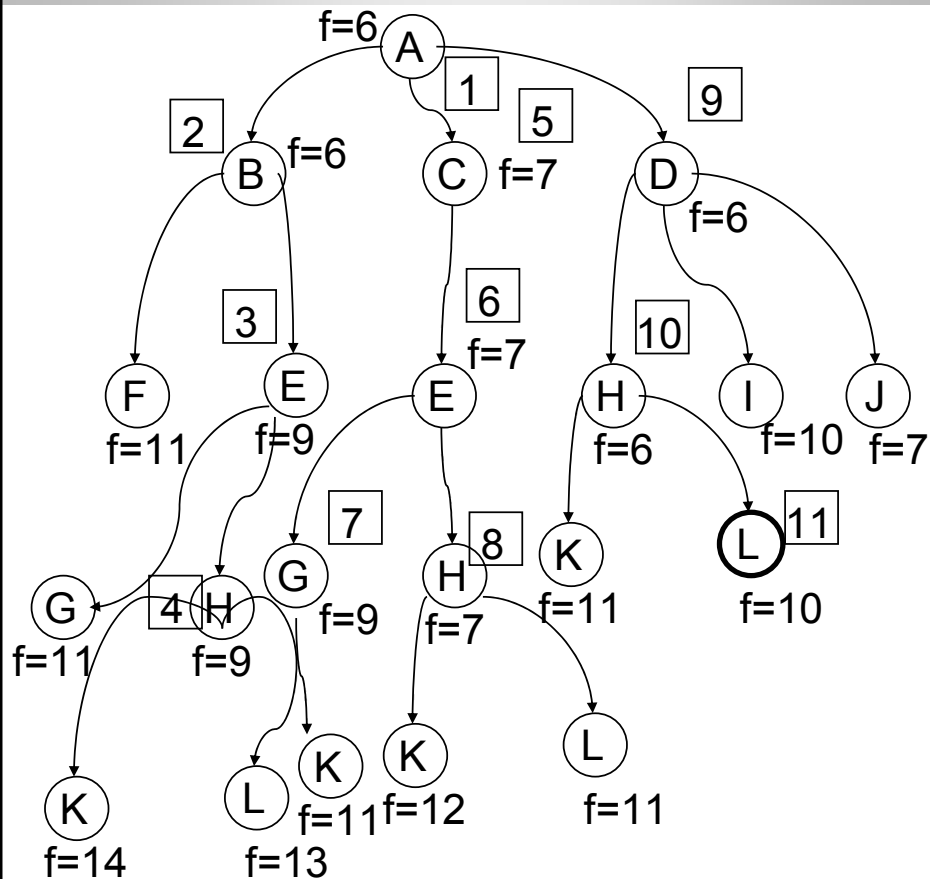


Ejemplo de aplicación de IDA\*, iteración  $f \leq 7$

Faltarían otras dos iteraciones:  $f \leq 9$ ,  $f \leq 10$



# Algoritmo IDA\*, III-d



Ejemplo de aplicación de IDA\*, iteración  $f \leq 10$

# Algoritmo IDA\*, IV

- IDA\* es completo y óptimo

- » necesita menos memoria (iterativo en profundidad)

- » Complejidad espacial:

$$O(b * \tilde{d}), \tilde{d} = \frac{f^*}{\text{minimo - valor - costes}}$$

- » Complejidad temporal:

- En problemas como el mapa de carreteras, cada iteración puede añadir sólo un nodo nuevo. Por tanto, si A\* expande N nodos, IDA\* necesitará N iteraciones y expandirá

$$1 + 2 + \dots + N = O(N^2)$$

- Una solución a este problema podría ser discretizar los posibles valores de  $C_k$  (múltiplos de  $\mathcal{E}$ ). Se tendrían heurísticas  $\mathcal{E}$ -admisibles.

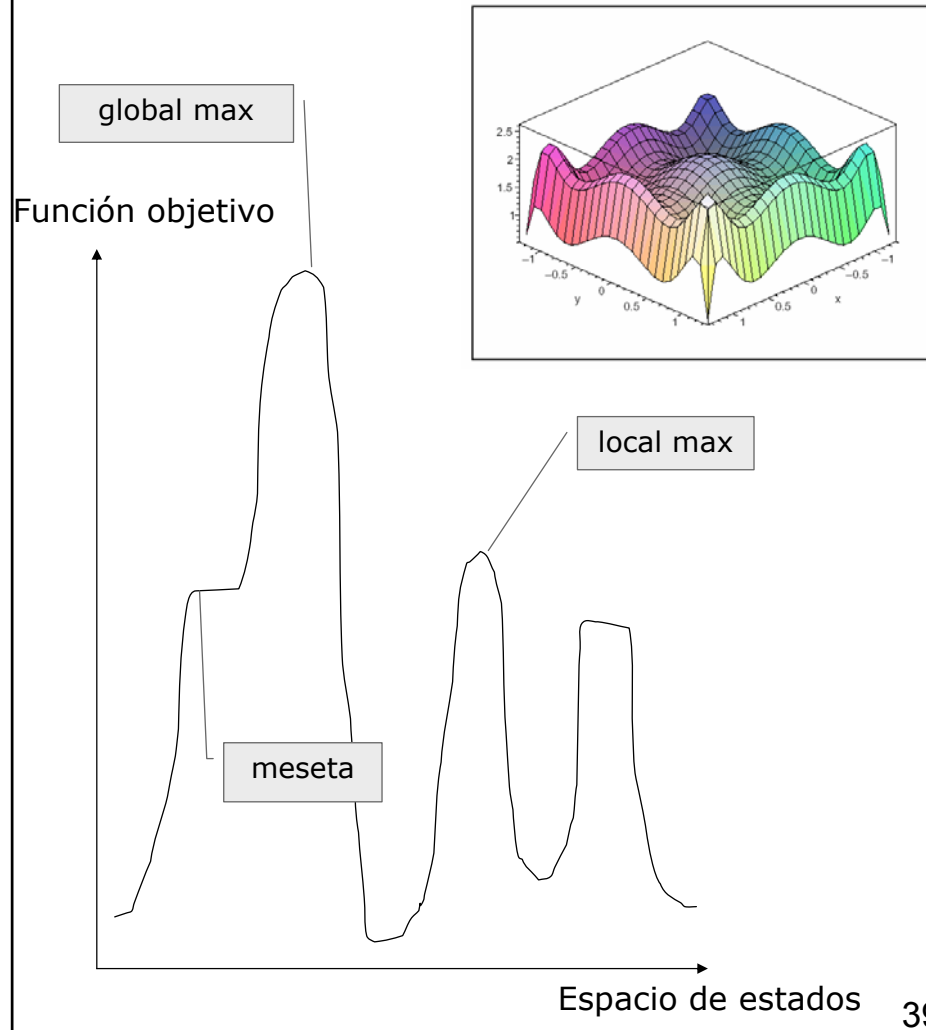
En tal caso, el número de

iteraciones sería:  $\frac{f^*}{\mathcal{E}}$

# Algoritmos de mejora iterativa, I

- Métodos basados en *teoría de la optimización*. Se trata de encontrar los puntos de una superficie cuyo valor de la función de evaluación (por ejemplo una heurística) sea máximo o mínimo.
- No es relevante el camino seguido para alcanzar el objetivo.
- Utilizan exclusivamente la información del estado actual y los de su entorno.
- Tienen dos ventajas primordiales:
  - » Utilizan muy poca memoria (generalmente constante)
  - » Encuentran soluciones en espacios de estados continuos (infinitos)
- Algoritmos
  - » Búsqueda con escalada (*Hill climbing* or *greedy local search*)
  - » Enfriamiento simulado (*simulated annealing*)
  - » Algoritmos genéticos, búsqueda tabú, redes neuronales, método hormiga, etc.

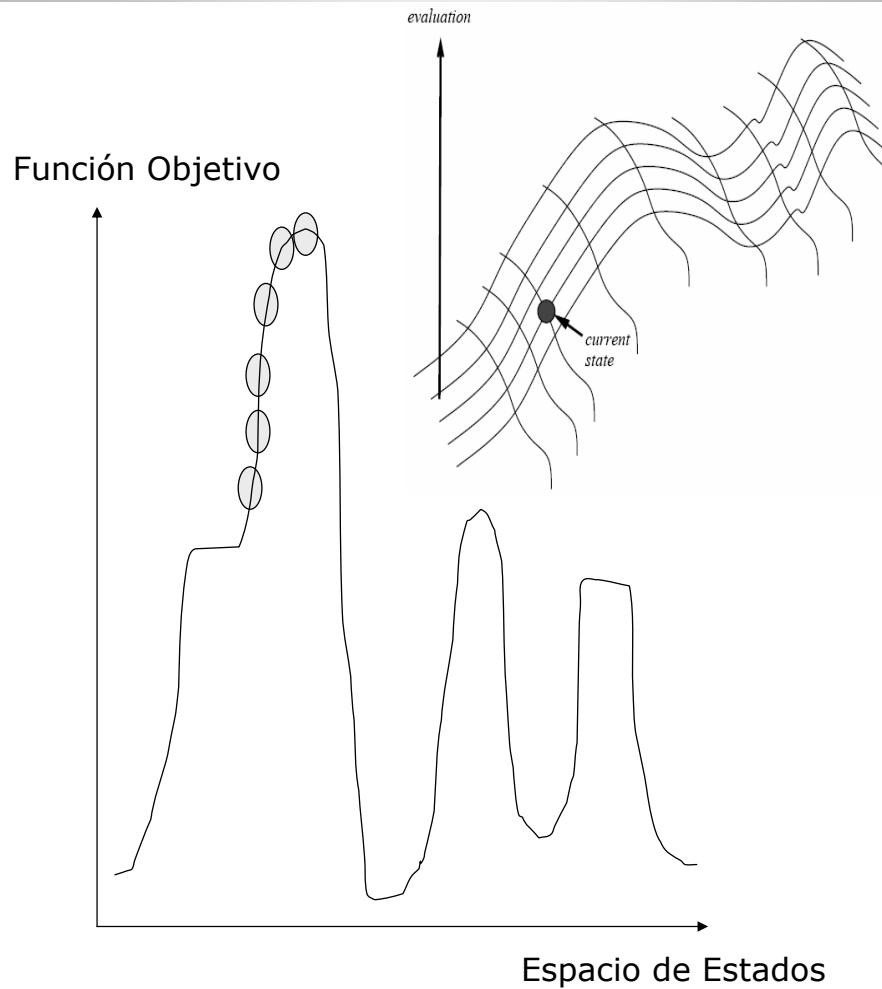
# Perfiles del espacio de estados



# Búsqueda con escalada, I

- **Búsqueda con escalada:**
  - » Consiste en un bucle que se desplaza continuamente en la dirección de crecimiento de valores (colina arriba).
  - » La condición de parada es encontrar un pico en cuyo entorno no existan valores mayores que el actual.
  - » No mantiene un árbol de búsqueda.
    - Los nodos sólo almacenan el estado actual y su valor objetivo.
  - » Si se puede elegir más de un sucesor que mejore el inicial (con el mismo valor de la función de evaluación), se elige al azar.
    - Estocástica
    - Primera elección
    - Reinicio aleatorio
  - » Inconvenientes:
    - Máximos locales
    - Zonas llanas (mesetas)
    - Crestas

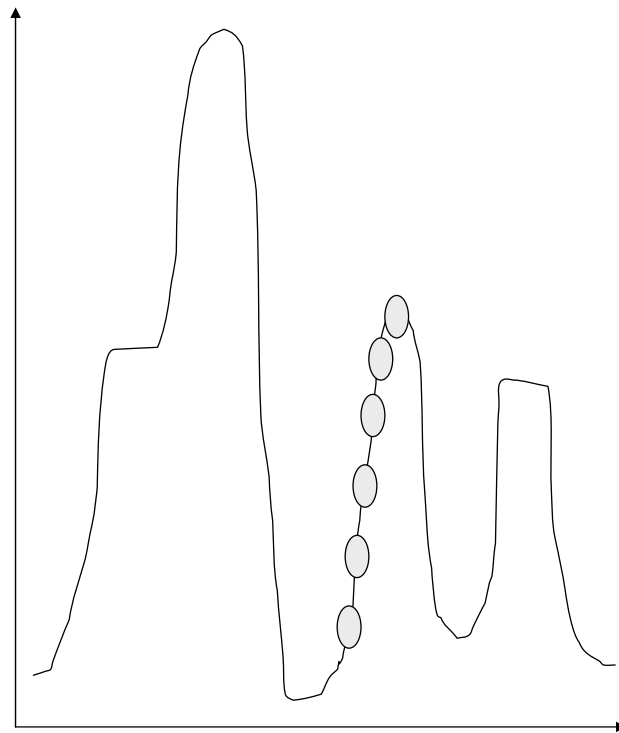
# Búsqueda con escalada, II



# Búsqueda con escalada, III

## Problema de los extremos locales

Función objetivo



Espacio de estados

# Búsqueda con escalada, IV

Bucle que, a partir de un estado, busca el estado vecino que aumente el valor de la función objetivo

Condición de parada: cuando encuentra un “pico”. Ningún estado vecino tiene valor mayor en la función con respecto al estado actual (si se utiliza h, entonces el vecino con h menor)

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
inputs: problem, a problem
local variables: current, a node
                 neighbor, a node

current ← MAKE-NODE(INITIAL-STATE[problem])
loop do
  neighbor ← a highest-valued successor of current
  if VALUE[neighbor] > VALUE[current] then return STATE[neighbor]
  current ← neighbor
```

## Inconvenientes:

- Máximos locales: se llega a un pico más bajo que el pico más alto del espacio de estados.
- Mesetas: áreas donde la función de evaluación es casi plana, no se puede progresar.
- Crestas: con pendientes laterales pronunciadas pero con una pendiente hacia el pico suave.

# Enfriamiento simulado, I

- **Enfriamiento simulado (*Simulated annealing*)**
  - » *Annealing*: “Proceso de enfriar lentamente un material en estado líquido hasta que se enfría en un estado cristalino de mínima energía”.
    - Si la temperatura se reduce de manera suficientemente lenta en un líquido, el material obtendrá su estado de más baja energía (ordenación perfecta).
- **Ejemplo:**
  - » Aplicación a CSPs como el de las N-damas:
    - Resolución de problema de 1.000.000 de damas en menos de 50 pasos.
      - Problemas de minimización de conflictos (en una columna al azar mover una dama a la casilla que cree menos conflictos)

R			2
		R	1
	R		2
			R

R		2	
		R	R
	R	2	
		1	

R	1		
	2		R
	R		
	2	R	

# Enfriamiento simulado, II

En vez de elegir el mejor movimiento, se elige un movimiento aleatorio. Si el movimiento mejora la situación ( $\Delta E > 0$ ), entonces es aceptado

El enfriamiento está planificado (*schedule[t]*)

Si no, el algoritmo acepta el movimiento con una probabilidad menor a 1, esta probabilidad se decreta exponencialmente con el empeoramiento de la evaluación  $\Delta E$

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
         schedule, a mapping from time to "temperature"
  local variables: current, a node
                 next, a node
                 T, a "temperature" controlling the probability of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
     $\Delta E$  ← VALUE[next] - VALUE[current]
    if  $\Delta E > 0$  then current ← next
    else current ← next only with probability  $e^{\Delta E/T}$ 
```

La probabilidad también se decreta con la temperatura. Inicialmente son permitidos "malos" movimientos cuando la temperatura es alta, y cada vez se permiten menos malos movimientos con el decremento de la temperatura.