Universidad
Francisco de Vitoria
UFV Madrid

Ingeniería del Conocimiento

Tema 4:
Búsqueda Informada (I)

Departamento de Ingeniería Informática
Universidad Francisco de Vitoria

Álvaro José García Tejedor
Curso 2018/19

Objetivos del tema

- Ubicación
  - Unidad 2: **BUSQUEDA EN ESPACIO DE ESTADOS**
  - *Tema 4: Búsqueda Informada: Heurísticas (I)*

- Objetivos generales
  - *Definir búsqueda informada* y entender su ámbito de aplicación en contraposición a la búsqueda ciega
  - Definir las *funciones heurísticas* y entender su uso en las búsquedas informadas evaluando estados en vez de explorar todos los caminos desde el estado inicial
  - Entender los algoritmos de búsqueda *Primero El Mejor*, especialmente el *Algoritmo A**, su uso y sus limitaciones
  - Saber *aplicar de cada método* en función de la completitud y *complejidad* espacial y temporal

Contenido

1. Introducción
2. Funciones heurísticas
3. Búsquedas "primero el mejor"
    1. Búsqueda avara
    2. Búsqueda A*
    3. Variaciones de A*
4. Búsquedas iterativas
    1. Hill Climbing
    2. Simulated Annealing

1. Introducción
2. Funciones heurísticas
3. Búsquedas "primero el mejor"
    1. Búsqueda avara
    2. Búsqueda A*
    3. Variaciones de A*
4. Búsquedas iterativas
    1. Hill Climbing
    2. Simulated Annealing

1. Introducción

- Búsqueda: exploración del espacio de estados por medio de la generación de sucesores de los estados explorados
  - Si se tiene conocimiento perfecto ➔ algoritmo exacto
  - Si no se tiene conocimiento ➔ búsqueda sin información
  - Los problemas reales están en posiciones intermedias

- Cuando se emplea información del espacio de búsqueda para evaluar el proceso y elegir que nodo del árbol de búsqueda es más prometedor para alcanzar la meta hablamos de estrategias de búsqueda
  - INFORMADAS
  - HEURÍSTICAS

  (*usan información disponible del problema*)

- La idea es utilizar una función de evaluación (heurístico) de cada nodo (del coste de llegar de él al estado final)

1. Introducción

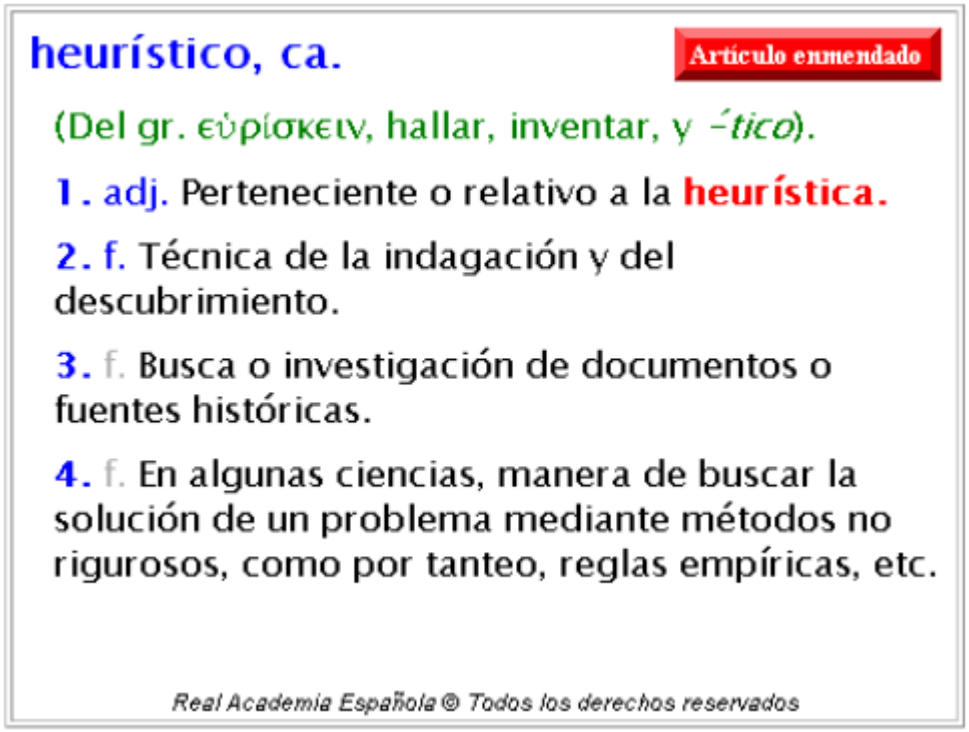
- Todos los problemas que trata la IA son NP-difíciles
  - Resolverlos exacta requiere búsqueda en un espacio de estados de tamaño exponencial.
    - No se sabe como evitar esa búsqueda.
    - No se espera que nadie lo consiga nunca.
  - Si existe un algoritmo rápido para resolver un problema, no consideramos que el problema requiera inteligencia

- Si un problema es NP-difícil y tenemos un algoritmo que encuentra la solución de forma rápida y casi siempre correcta, podemos considerar que el algoritmo es "inteligente".
  - Implica que la búsqueda está sujeta a error

1. Introducción
2. Funciones heurísticas
3. Búsquedas "primero el mejor"
    1. Búsqueda avara
    2. Búsqueda A*
    3. Variaciones de A*
4. Búsquedas iterativas
    1. Hill Climbing
    2. Simulated Annealing

2. Funciones heurísticas

- Heurística (¡Eureka!):



heurístico, ca. Artículo enmendado

(Del gr. εὑρίσκειν, hallar, inventar, y *-tico*).

1. adj. Perteneciente o relativo a la **heurística.**

2. f. Técnica de la indagación y del descubrimiento.

3. f. Busca o investigación de documentos o fuentes históricas.

4. f. En algunas ciencias, manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc.

- *Técnica o regla empírica que ayuda a encontrar la solución de un problema (pero que no garantiza que se encuentre)*
- *Criterios, métodos o principios para decidir cuál de entre varias acciones promete ser la mejor para alcanzar una determinada meta.*

2. Funciones heurísticas

- Son características de los métodos heurísticos:
  - No garantizan que se encuentre una solución, aunque existan soluciones (sacrifica la completitud).
  - Si encuentran una solución, no se asegura que sea la mejor (longitud mínima o de coste óptimo).
  - En algunas ocasiones (que, en general, no se podrán determinar a priori) encontrarán una solución aceptablemente buena en un tiempo razonable.

- Se representan mediante
  - Funciones *h(n)*
  - Metareglas

- Las heurísticas se descubren resolviendo modelos simplificados (*relajados*) del problema real

2. Funciones heurísticas

- Asocia a cada estado del espacio de estados una cierta cantidad numérica que evalúa de algún modo lo prometedor que es ese estado para alcanzar un estado objetivo
- ¿Qué es "mejor" valor heurístico?
  - Si estimamos la "calidad" de un estado
    - Los estados de mayor valor heurístico son los preferidos
  - Si estimamos lo próximo que se encuentra de un estado objetivo (coste estimado del camino más barato)
    - Los estados de menor valor son los preferidos
- Ambos puntos de vista son complementarios
- Convenio: asumiremos la 2ª interpretación:
  - Valores no negativos
  - El mejor es el menor
  - Los objetivos tienen valor heurístico 0

2. Funciones heurísticas

Ejemplo: 8-puzzle

- Restricciones:
  1. Una ficha solo se puede mover al hueco
  2. Una ficha solo se puede mover a casillas adyacentes horizontales o verticales
  3. En cada paso, se intercambian los contenidos de dos casillas

- Relajaciones:
  - Si quitamos 1, 2 y 3, heurística $\mathbf{h_1}$:
    número de casillas mal colocadas respecto al objetivo
    - Es la heurística más sencilla y parece bastante intuitiva
    - No usa información relativa al esfuerzo (nº de movimientos) necesario para llevar una ficha a su sitio

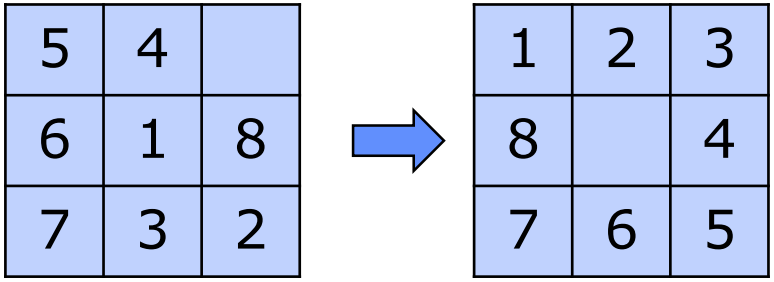
2. Funciones heurísticas

- Si quitamos 1, heurística $\mathbf{h_2}$:

  suma de las distancias de las fichas a sus posiciones en el objetivo
  - Como no hay movimientos en diagonal, se sumarán las distancias horizontales y verticales
  - Distancia de Manhattan distancia taxi: número de cuadros desde el sitio correcto de cada cuadro *excluyendo la vacía*

$$|(x_f - x_i)| + |(y_f - y_i)|$$

- Ejemplo



- Solución
  - h_1: 8
  - h_2: 2+3+3+2+4+2+0+2 = 18

2. Funciones heurísticas

- Sin embargo, estas dos heurísticas no dan importancia a la dificultad de la inversión de fichas
  - Si 2 fichas están dispuestas de forma contigua y han de intercambiar sus posiciones, ponerlas en su sitio supone (bastante) más de 2 movimientos
  - Heurística $\boldsymbol{h_3}$: *doble del nº de pares de fichas a "invertir entre sí"*
    - Tampoco es buena porque se centra solo en un cierto tipo de dificultad sin considerar el problema general
    - En particular, tendrán valor 0 muchos estados que no son el objetivo

- Se suelen usar heurísticas compuestas

$$\boldsymbol{h_4 = h_2 + h_3}$$

  - Mejor heurística pero requiere más cálculo

2. Funciones heurísticas

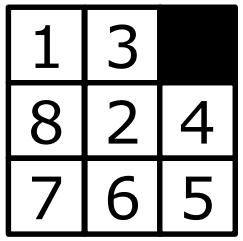
Heurística dominante

- $h_i(n)$ es dominante si para un problema dado

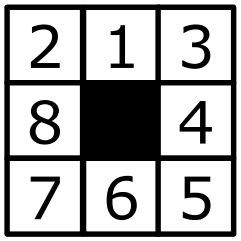
$$h_i(n) \geq h_h(n) \ \forall h \ \forall n$$

- Si $h_2 \geq h_1 \ \forall n$ ➔ h_2 (domina a/esta mas informada que) h_1
  - La dominación se traduce en eficiencia: un heurístico dominante expande menos nodos
  - La distancia de Manhattan esta mas informada que la heurística de numero de casillas mal colocadas

2. Funciones heurísticas



Estado objetivo



$h_1 = 2$
$h_2 = 2$
$h_3 = 0$

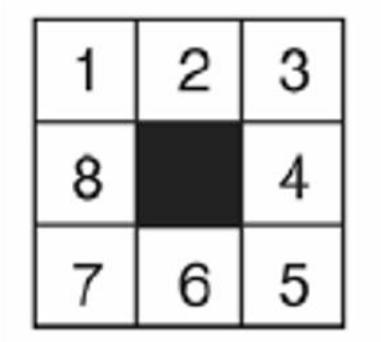
$h_1 = 2$
$h_2 = 2$
$h_3 = 2$

$h_4 = 2$ → Dominante ← **$h_4 = 4$**

Reales = 2

Reales > 10

2. Funciones heurísticas



Estado objetivo

	H1	H2	H3	H4
	6	6	0	6
	3	4	0	4
	6	6	0	6

2. Funciones heurísticas

- En general, los métodos heurísticos son preferibles a los métodos no informados en la solución de problemas difíciles para los que una búsqueda exhaustiva necesitaría un tiempo demasiado grande.

- *Es necesario un compromiso entre el coste de la función heurística y la mejora que supone en la búsqueda*

- *Es preferible usar una función heurística dominante siempre y cuando sea admisible*

2. Funciones heurísticas

- Ejemplo: el coste de resolver el puzzle de las 8 piezas mediante las estrategias de búsqueda por profundización iterativa y A* con heurísticos h_1 y h_2

  - $d = 14$
    - profundización iterativa → 3.473.941 nodos
    - A* con $h1$ → 539 nodos
    - A* con $h2$ → 113 nodos

  - $d = 24$
    - profundización iterativa → !demasiados nodos!
    - A* con $h1$ → 39.135 nodos
    - A* con $h2$ → 1.641 nodos

2. Funciones heurísticas

- Búsquedas heurísticas o informadas

  - Primero el mejor (PEM o *best-first*)
    - Búsqueda avara/voraz (*greedy search*)
    - *Búsqueda A**
    - Variaciones de A*
  - Mejora iterativa
    - Métodos de gradiente (*hill-climbing*)
    - *Simulated annealing*
  - Búsqueda con adversarios
    - *Búsqueda MiniMax con decisiones imperfectas*
    - *Poda Alfa-Beta*
  - Búsqueda con restricciones

1. Introducción
2. Funciones heurísticas
3. Búsquedas "primero el mejor"
    1. Búsqueda avara
    2. Búsqueda A*
    3. Variaciones de A*
4. Búsquedas iterativas
    1. Hill Climbing
    2. Simulated Annealing

3. Búsquedas "primero el mejor"

- Se utiliza una función de evaluación *f(n)* para cada nodo y se expande el nodo mejor evaluado no expandido
  - Misma idea que en la búsqueda de coste uniforme:
    - Cola con prioridad, mantiene continuamente la frontera con orden creciente de *f(n)*
    - Se expande el nodo que *parece* mejor según *f(n) (aunque es una función inexacta)*

- La función heurística *h(n)* es el coste estimado del camino más barato desde n al objetivo
  - Condición: Si n es un nodo objetivo entonces *h(n)* = 0

- Válido cuando lo que interesa es encontrar el camino completo, no la solución en si.

3. Búsquedas "primero el mejor"

- Tipos de búsquedas PEM:

  - Búsqueda en anchura
    - f(n) = profundidad(n) **mínima**
  - Búsqueda de coste uniforme (Dijkstra)
    - f(n) = g(n) **mínima**

  Ya hemos visto estas búsquedas NO HEURISTICAS al ver los métodos NO INFORMADOS

  - Búsqueda voraz o avara (*greedy search*)
    - f(n) = h(n) **mínima**
  - Búsqueda A*
    - f(n) = g(n)+h(n) **mínima**
  - Variaciones del A* que funcionan acotando el uso de memoria

3.1 Búsqueda avara

- *f(n)* estima el coste del nodo n hasta la meta, con lo que se expande el nodo *NO EXPANDIDO* que *parece* estar más cerca de la meta
- *f(n)* es directamente la función heurística *h(n)* del estado

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

  - *h* puede ser cualquier función siempre y cuando $h(n) = 0$ en los nodos que representan estados objetivo

- Propiedades:
  - Completa: **No**, en general. **Si**, si se aplican políticas de poda de bucles.
  - Complejidad tiempo: $O(b^d)$ hay que recorrer todos los nodos
  - Complejidad espacio: $O(b^d)$ hay que recorrer todos los nodos
  - Optima: **No**

3.1 Búsqueda avara

- La búsqueda voraz:
  - es propensa a comienzos erróneos
  - como la búsqueda primero en profundidad, no es completa ni óptima
    - prefiere seguir un camino hasta el final
    - puede atascarse en bucles infinitos

- Las complejidades temporal y espacial pueden reducirse sustancialmente con un buen heurístico.

¡¡Una mala heurística es peor que una mala búsqueda!!

3.1 Búsqueda avara

```
PROCEDIMIENTO VORAZ(Estado-inicial, Estado-Final)

    *ABIERTO* = *ESTADO-INICIAL*
    Hacer *CERRADO* vacío

    BUCLE (Repetir el proceso mientras ABIERTO ≠ vacío)
       1. NODO-ACTUAL = EXTRAE-PRIMERO(ABIERTO)
       2. Poner NODO-ACTUAL en *CERRADO*
       3. Si ES-ESTADO-FINAL(ESTADO(NODO-ACTUAL))
                devolver CAMINO(NODO-ACTUAL)
       4. Si no, FUNCION SUCESORES(NODO-ACTUAL)
                GESTIONA-COLA(ABIERTO,SUCESORES)
                Si no están en *ABIERTO* o *CERRADO* añadir *SUCESORES*
                ordenadamente a ABIERTO en orden creciente de h(n)
    FIN DE BUCLE

    Devuelve FALLO ☹
```

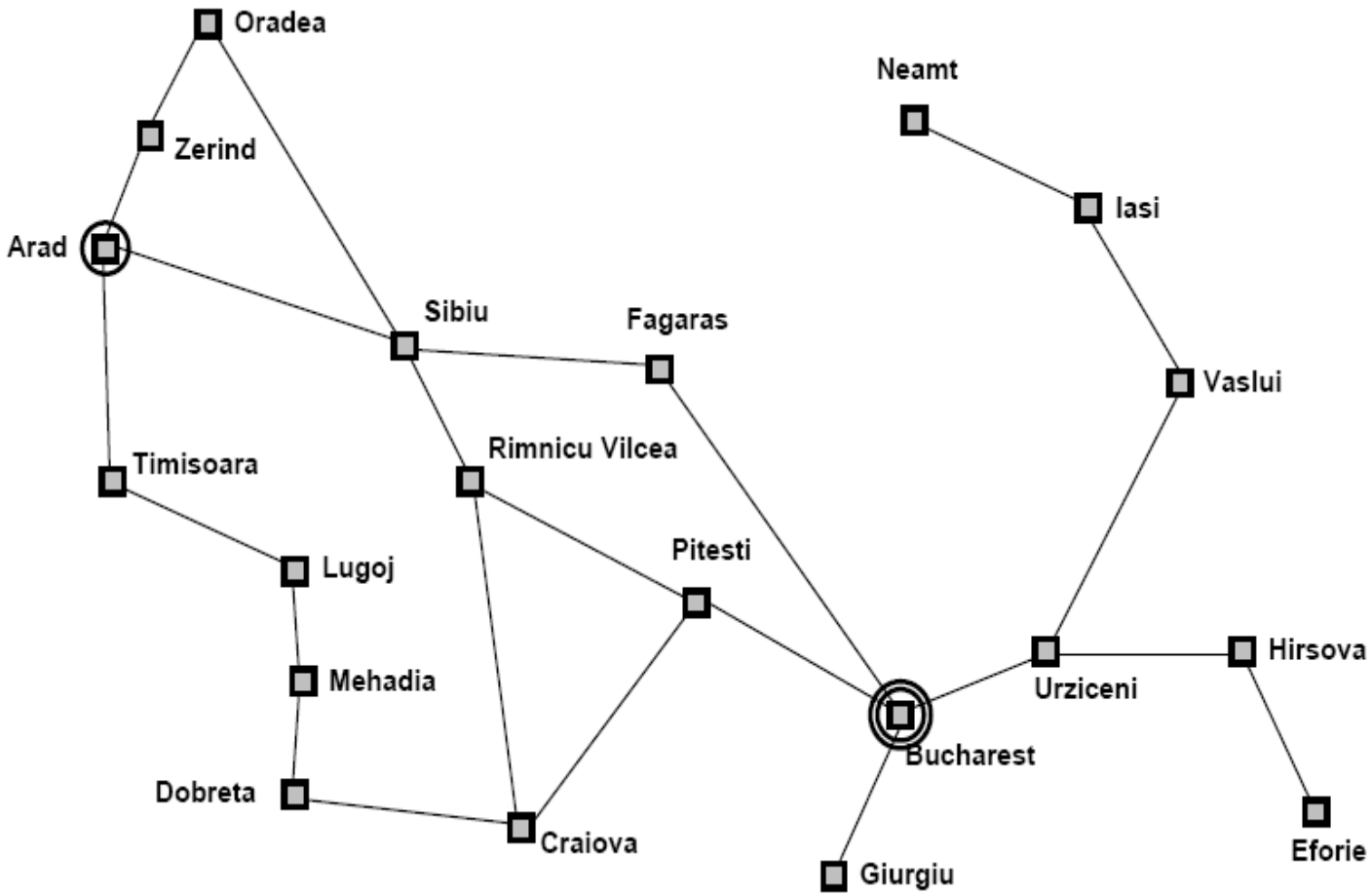
3.1 Búsqueda avara

- El problema del viaje por Rumanía.
  - Estado inicial: estamos en una ciudad.
  - Estado meta: quiere viajar a otra ciudad por la mejor ruta posible (la más corta)
  - Medios: Las ciudades colindantes están unidas por carreteras; se dispone de un mapa con la disposición de las provincias y sus "coordenadas" en kilómetros respecto al "centro"

- *F(n)*: asignar a cada nodo la distancia aérea (en línea recta) con el estado objetivo (distancia euclídea entre las coordenadas de dos ciudades).
  - Se elige una ciudad como siguiente en el camino cuando la distancia aérea a la meta sea la menor.

$$f(n) = h(n) \text{ mínima}$$

3.1 Búsqueda avara



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

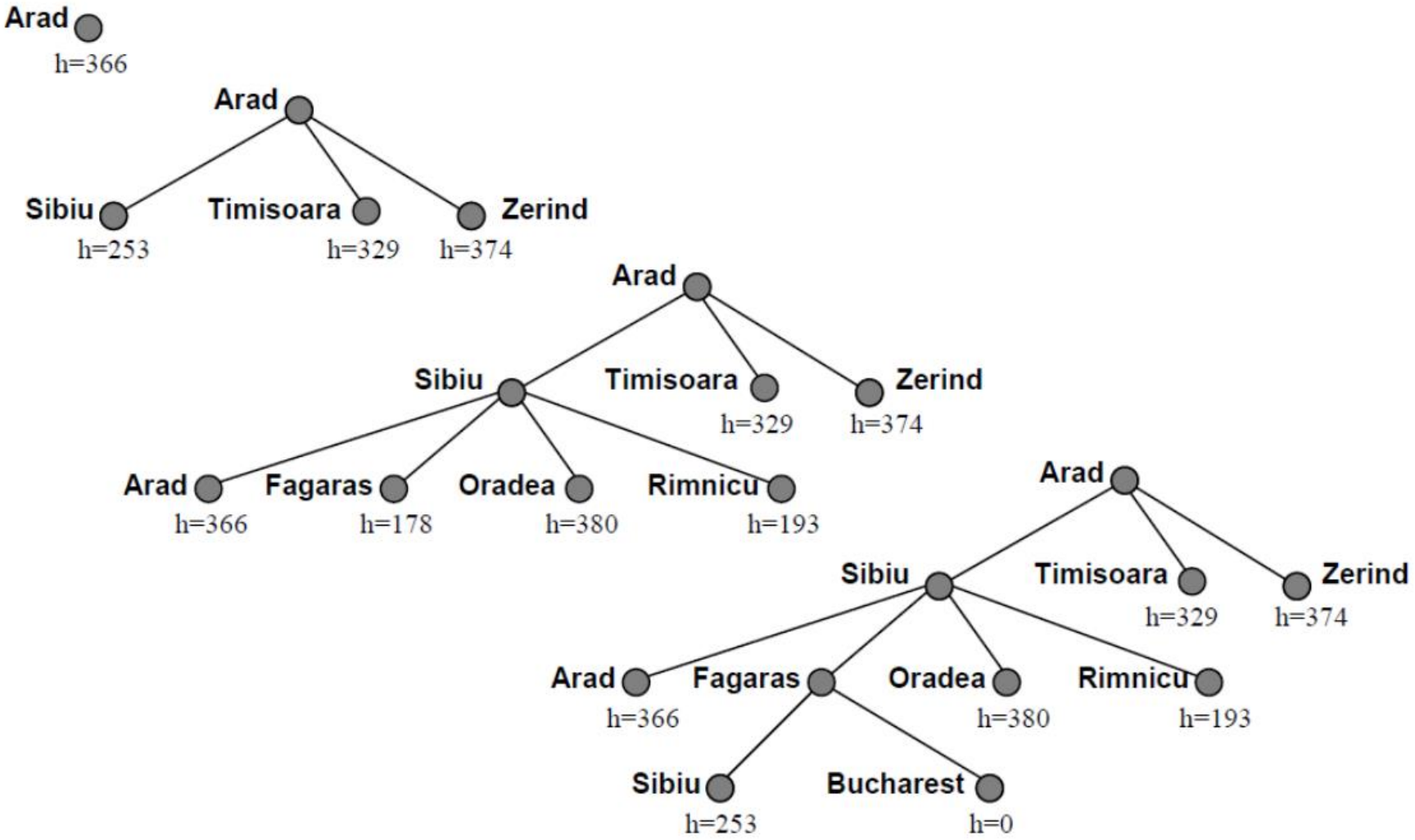
Distancias a Bucarest en Km (línea recta)

3.1 Búsqueda avara

- Para el problema de hallar una ruta entre Arad y Bucarest, la búsqueda voraz con el heurístico $h_{DLR}(n)$:
  - encuentra una solución sin expandir ningún nodo que no este incluido en la misma (coste de búsqueda mínimo),
  - aunque la solución no es óptima

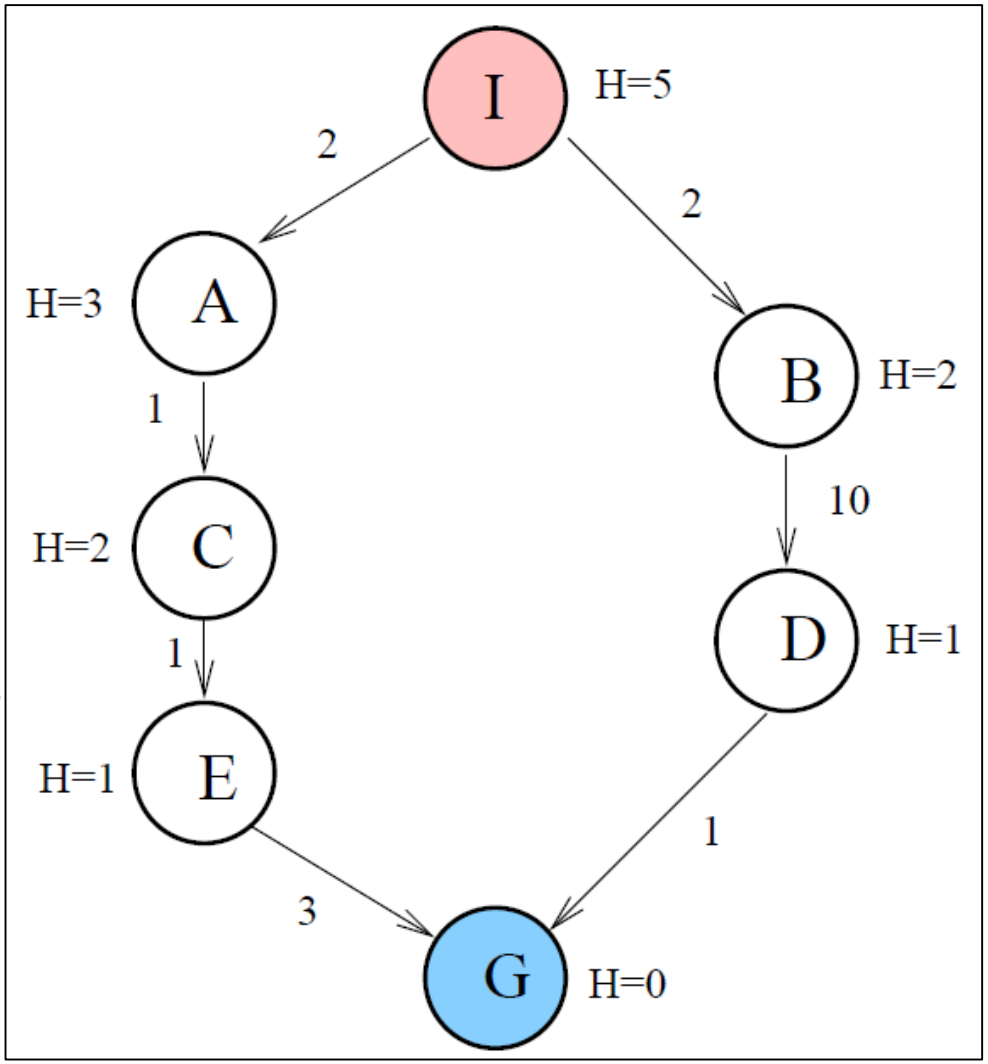
- $h_{DLR}(n)$: (Distancia en Línea Recta)
  - necesita de las coordenadas de las ciudades del mapa
  - es útil porque sabemos que las carreteras entre dos ciudades tienden a ser rectas (conocimiento especifico del problema)

3.1 Búsqueda avara



3.1 Búsqueda avara

- PROBLEMA: NO se encuentra la solución óptima
  - Solución encontrada por búsqueda voraz: I-B-D-G
  - Causa: no se han tenido en cuenta *los costes* de los caminos ya recorridos



3.2 Búsqueda A*

- Hart, Nilsson y Raphael, 1968
- Búsqueda A*
  - Evita expandir caminos que ya son muy costosos minimizando el costo estimado total de la solución.
  - Combina:
    - la búsqueda voraz, que minimiza el coste al objetivo $h(n)$
      - Búsqueda en profundidad
    - la búsqueda de coste uniforme, que minimiza el coste acumulado $g(n)$
      - Búsqueda en anchura

- Expande primero el nodo no expandido más prometedor hasta ese momento según la Función de Evaluación:

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

3.2 Búsqueda A*

- Donde
  - *f(n)*: función de evaluación
    - Coste estimado total hasta la meta pasando por n.
    - Expresa el mínimo estimado de la solución que pasa por el nodo n
  - *g(n)*: función de coste para ir desde el nodo inicial al actual
  - *h(n)*: función heurística que mide la distancia (o coste) ***estimada*** desde *n* a algún nodo meta

- Los valores reales solo se conocen al final de la búsqueda
  - *h*(n)*: coste real para ir del nodo *n* a algún *nodo meta*
  - *f*(n)*: coste real para ir del *nodo inicial* a algún *nodo meta* a través de *n*
  - *g(n)* si se conoce y se calcula como la suma de los costes de los arcos recorridos, $k(n_i,n_j)$

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

3.2 Búsqueda A*

Heurística admisible

- Como no conocemos el valor h^* usamos una función de estimación a la que llamamos $h_i(n)$ que será admisible si

$$h^*(n) \geq h(n) \ \forall n$$

- *El coste de una solución óptima en un problema relajado es una heurística admisible para el problema original*

- Pero si h no es admisible, entonces A* es simplemente A

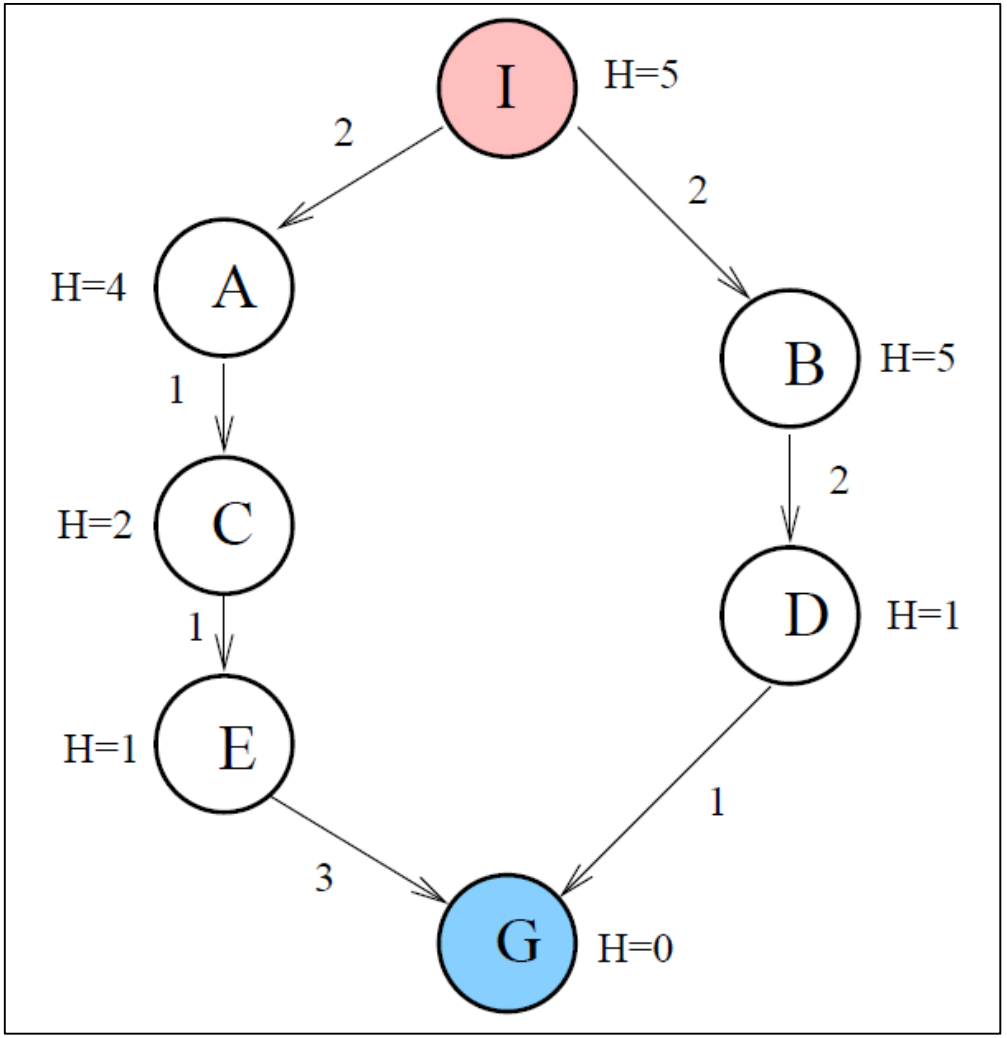
(esto es importante)

3.2 Búsqueda A*

- La heurística controla el comportamiento de A*.
  - $h(n) = 0$ $\forall n$: ***Búsqueda Dijkstra***. Solamente $g(n)$ importa.
    - Garantiza encontrar el camino más corto
  - $h(n) \leq h^*(n)$ $\forall n$: ***Búsqueda A****. $h(n)$ admisible.
    - ***Optimista***: Subestima el coste real de llegar al objetivo
    - Garantiza encontrar el camino más corto.
    - Cuanto menor sea $h(n)$, más nodos expande A*, haciéndolo más lento. Por eso interesa que h sea dominante (***lo más alta posible pero admisible***)
  - $h(n) = h^*(n)$ $\forall n$: ***Búsqueda A****.
    - Seguirá el mejor camino y ***nunca expandirá ningún otro nodo***, por lo que A* es muy rápido.
  - $h(n) > h^*(n)$ para algún n: ***Búsqueda A***. $h(n)$ no admisible.
    - No está garantizado que A* encuentre el camino más corto, pero el algoritmo puede correr más rápido.
  - $h(n) >> g(n)$ $\forall n$: ***Búsqueda Avara***. Solamente $h(n)$ importa

3.2 Búsqueda A*

- PROBLEMA: NO
  se encuentra la solución óptima
  - Solución encontrada por A*:
    I-A-C-E-G
  - Causa: la heurística *sobrestima* el coste real en B



3.2 Búsqueda A*

Heurística consistente

- Una heurística *h(n)* es consistente si

$$h(padre) \leq h(hijo) + k(padre, hijo)\ \forall n$$

consistente ➔ admisible

- Si *h(n)* es admisible y consistente entonces *f(n)*, a lo largo de cualquier camino, no disminuye (es *monótona no decreciente*)

3.2 Búsqueda A*

- En el nodo inicial,
  - *g(inicial)*=0

    f(inicial) = h(inicial)

    f(inicial) = h*(inicial)*
  - Como además *h(inicial) ≤ h*(inicial)*

    f(inicial) ≤ f(inicial)*

- En el nodo final,
  - *h(final)*=0

    f(final) = g(final)
  - *Como además h*(inicial) = g(final)*

    f(final) = f(inicial)*

- Luego

  f(inicial) ≤ f(final)

3.2 Búsqueda A*

- La secuencia de nodos expandidos por A* estará en orden no decreciente de f(n)
  - El primer nodo expandido cada vez debe ser una solución óptima, ya que todos los posteriores serán al menos tan costosos
  - No revisita nodos. La primera expansión es la mejor

- Si f^* es el coste de la solución óptima, entonces
  - A* expande todos los nodos con $f(n) < f^*$
  - A* puede expandir algunos nodos situados sobre "*la curva de nivel objetivo*" (donde $f(n) = f^*$) antes de seleccionar un nodo objetivo
  - A* no expande ningún nodo con $f(n) > f^*$ (poda)

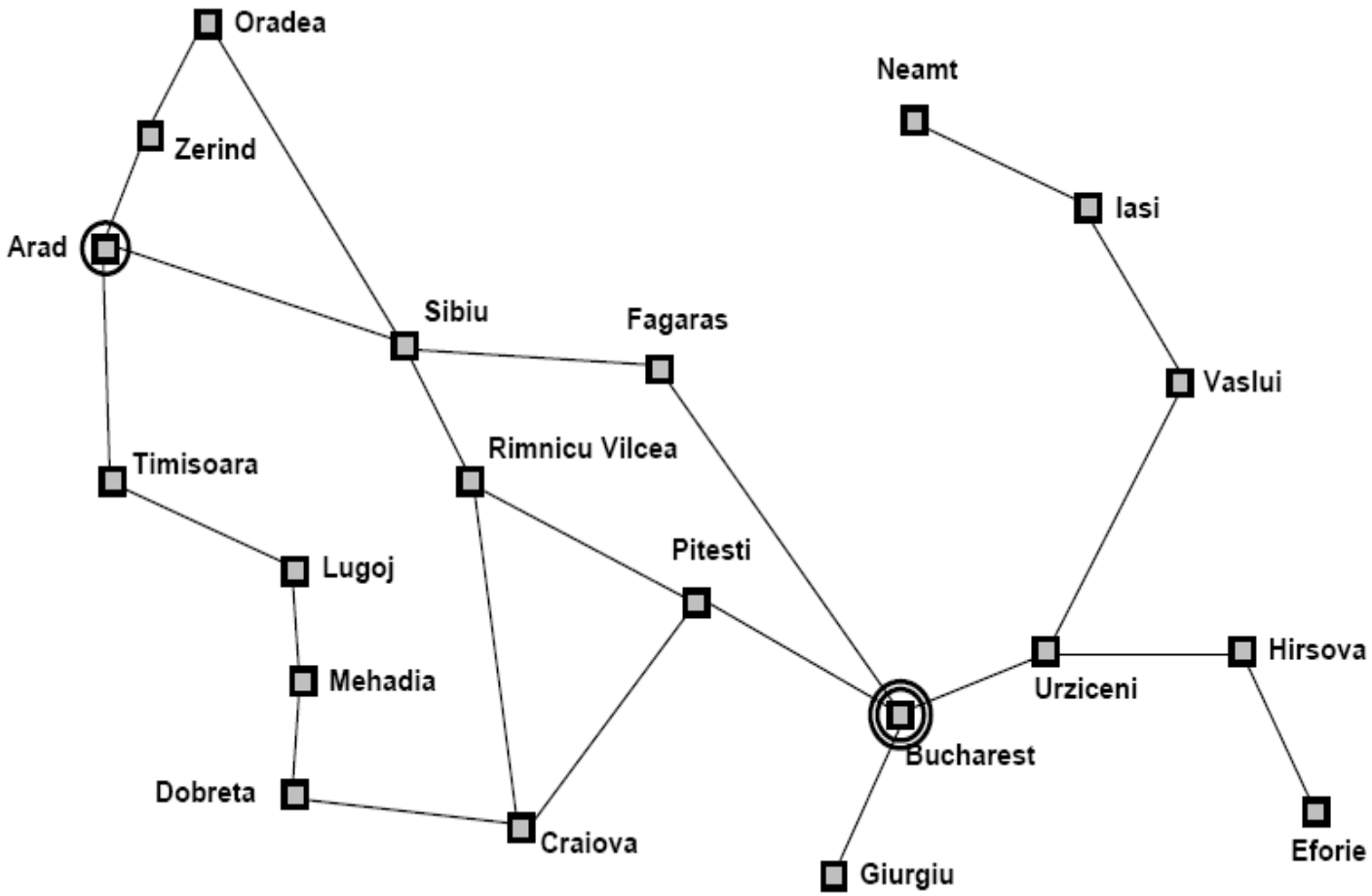
3.2 Búsqueda A*

El problema del viaje por Rumanía:

- *F(n)*: asignar a cada nodo la distancia desde el origen + la distancia en línea recta al estado objetivo.
  - Se elige una ciudad como siguiente en el camino cuando la suma de la distancia por carretera a la ciudad actual más la distancia aérea a la meta sea la menor.

$$f(n) = g(n)+h(n) \text{ mínima}$$

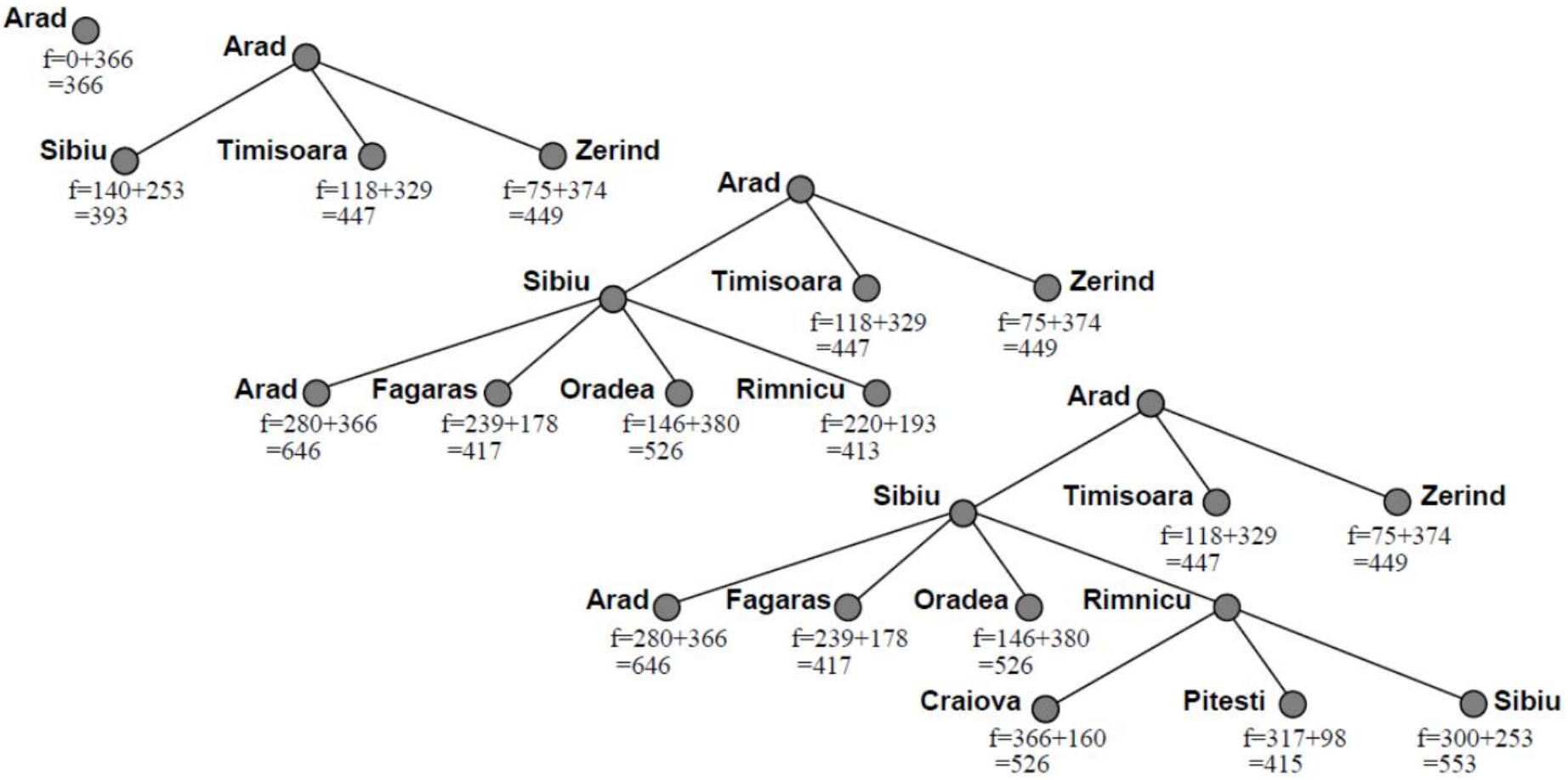
3.2 Búsqueda A*



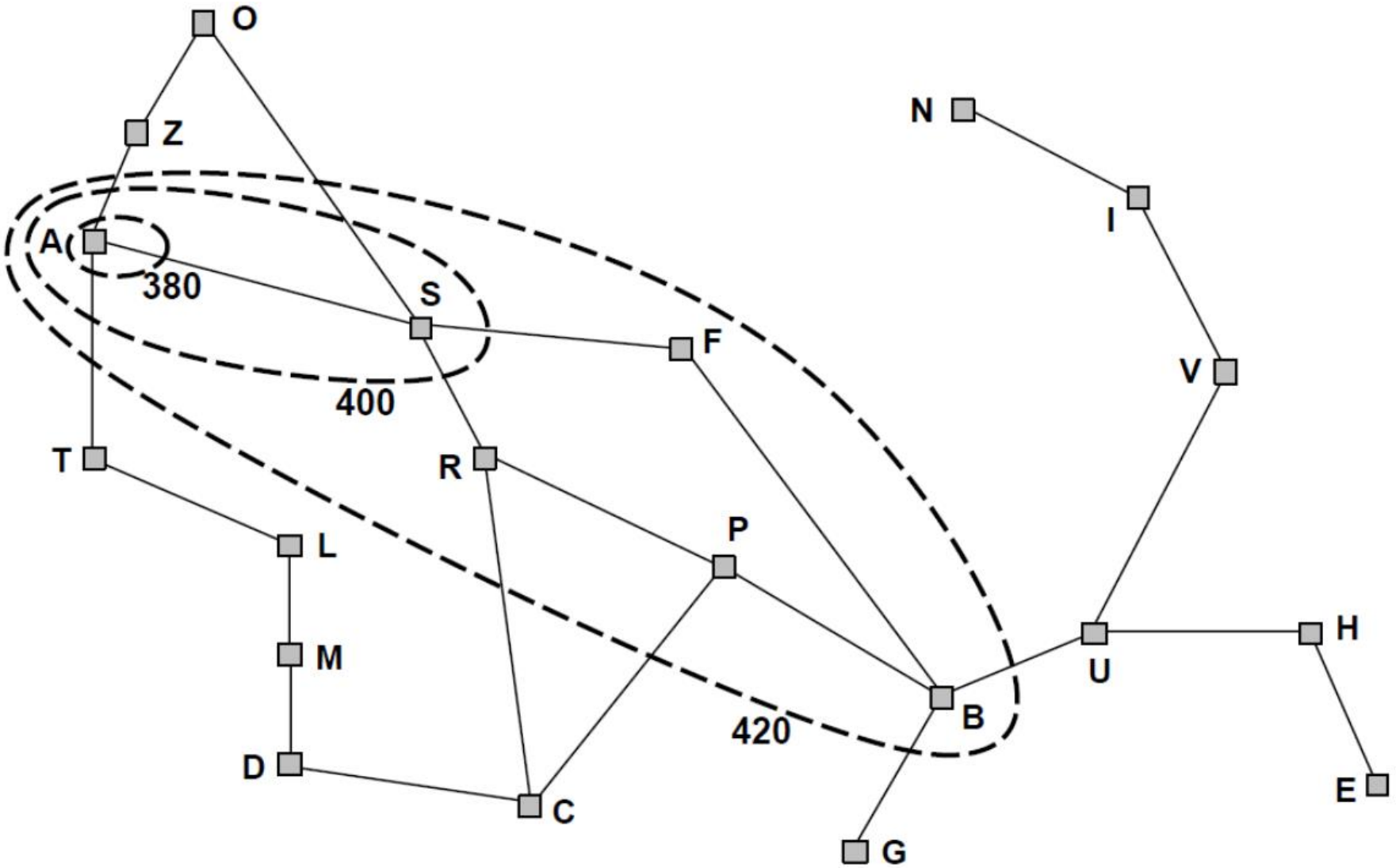
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Distancias a Bucarest en Km (línea recta)

3.2 Búsqueda A*



3.2 Búsqueda A*



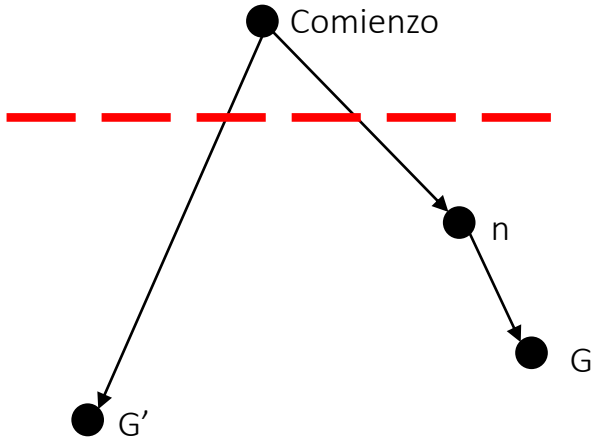
3.2 Búsqueda A*

- Si …
  - *h* es admisible
  - El número de sucesores de n (b) es finito $\forall n$
  - $k(n_i,n_j) \geq 0$ en todo arco

- entonces A* es:
  - Completa: **Si**, si existe solución la encuentra
  - Complejidad tiempo: $O(b^d)$
  - Complejidad espacio: $O(b^d)$
  - Optima: **Si**
  - A* es además óptimamente eficiente
    - Ningún otro algoritmo óptimo expande menos nodos para cualquier heurístico
    - Lo hemos visto al discutir la consistencia de las heurísticas

3.2 Búsqueda A*

- Supongamos que se ha generado un objetivo subóptimo (*G'*) y que está en la cola
- Sea *n* un nodo no expandido en el camino más corto al objetivo óptimo *G*



f(G') = *g(G')* ya que *h(G')= 0*
f(G) = *g(G)* ya que *h(G)= 0*
f(G') > *f(G)* ya que *G'* no es óptimo
f(G') ≥ *f(n)* ya que *h* es consistente

Por lo que A* no expandirá G' antes de alcanzar G

3.2 Búsqueda A*

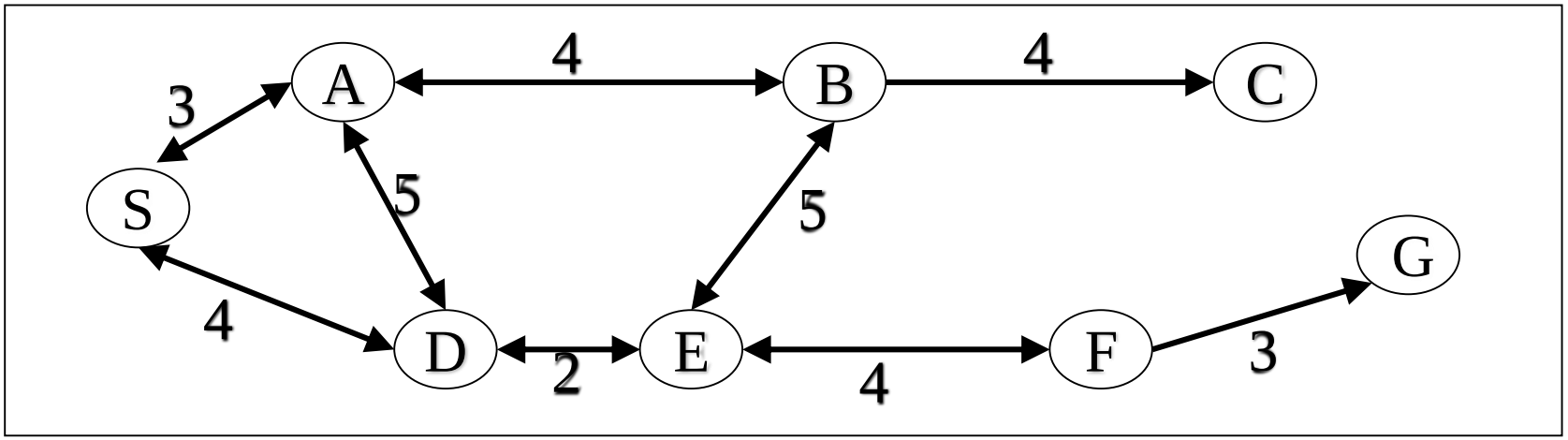
```
PROCEDIMIENTO A-STAR(Estado-inicial, Estado-Final)

    *ABIERTO* = *ESTADO-INICIAL*
    Hacer *CERRADO* vacío

    BUCLE (Repetir el proceso mientras ABIERTO ≠ vacío)
       1. NODO-ACTUAL = EXTRAE-PRIMERO(ABIERTO)
       2. Poner NODO-ACTUAL en *CERRADO*
       3. Si ES-ESTADO-FINAL(ESTADO(NODO-ACTUAL))
                devolver CAMINO(NODO-ACTUAL)
       4. Si no, FUNCION SUCESORES(NODO-ACTUAL)
                5. Si SUCESOR ya está en *CERRADO*,
                        Si g(SUCESOR) es menor, insertar ordenadamente en *ABIERTO*
                                Actualizar coste y camino
                6. Si SUCESOR ya está en *ABIERTO*,
                        Si g(SUCESOR) es menor, actualizar coste, posición y camino
                7. GESTIONA-COLA(ABIERTO,SUCESORES) Añadir *SUCESORES* a *ABIERTO*
                   en orden creciente de f(n)
    FIN DE BUCLE
    Devuelve FALLO ☹
```

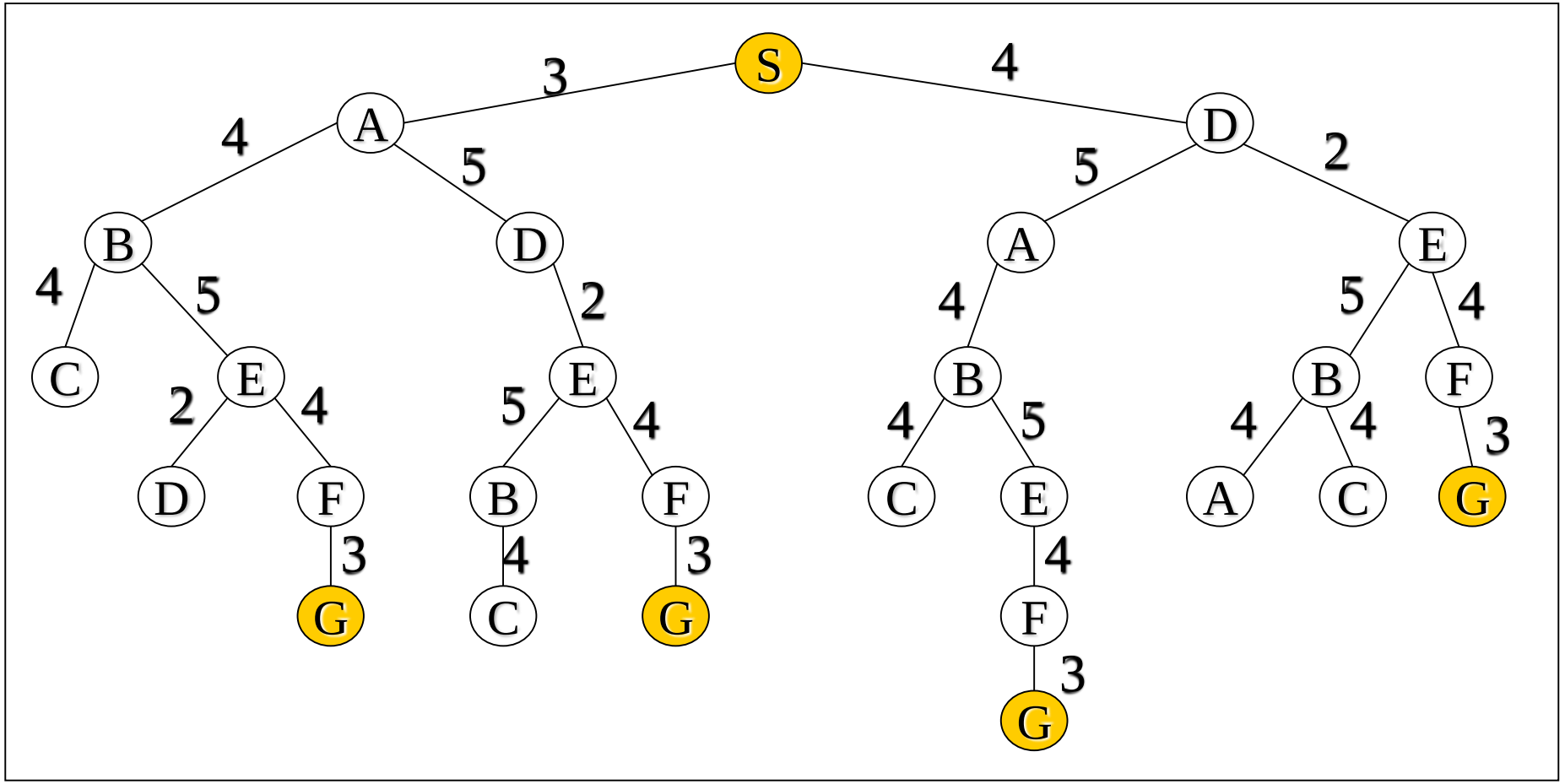
3.2 Búsqueda A*

- Ejemplo: sea el siguiente grafo



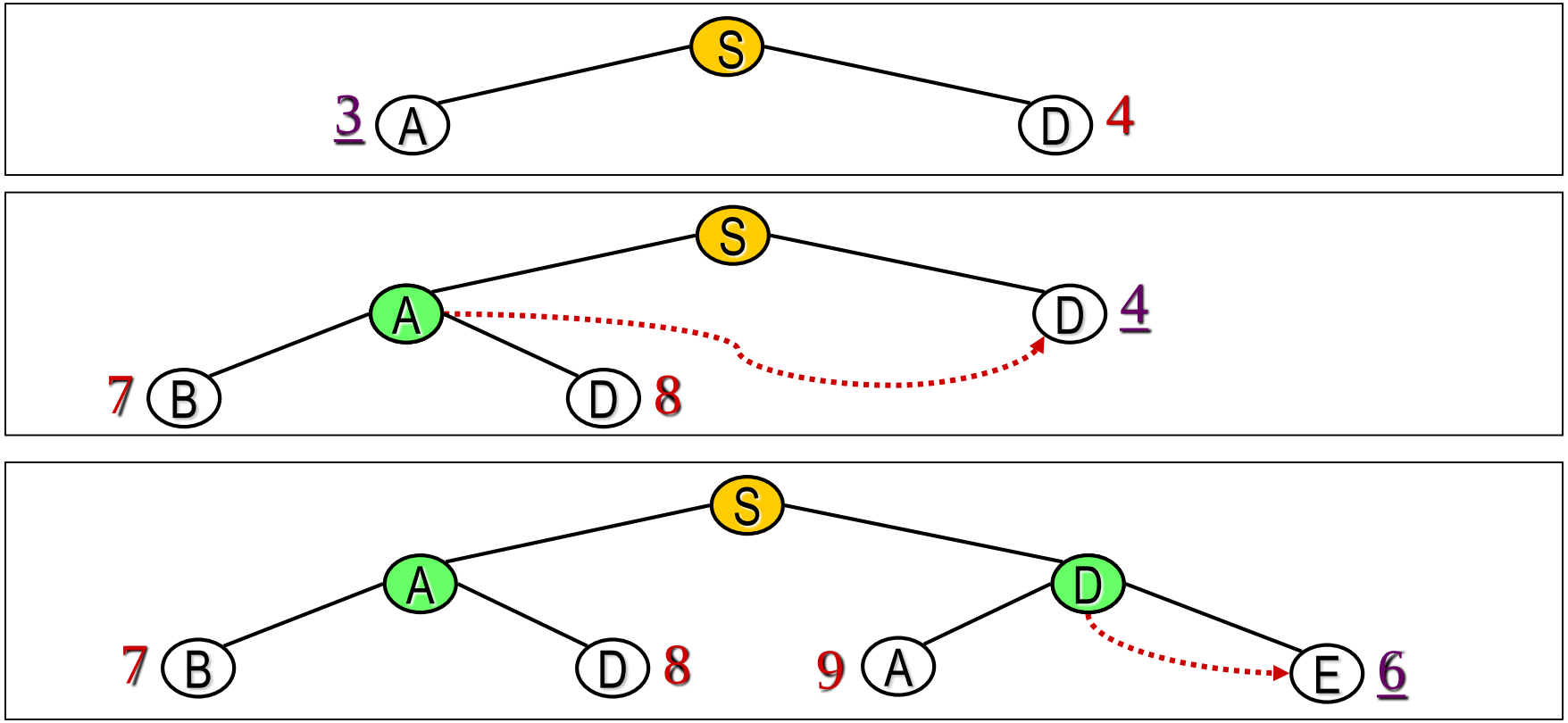
3.2 Búsqueda A*

- Que genera este árbol

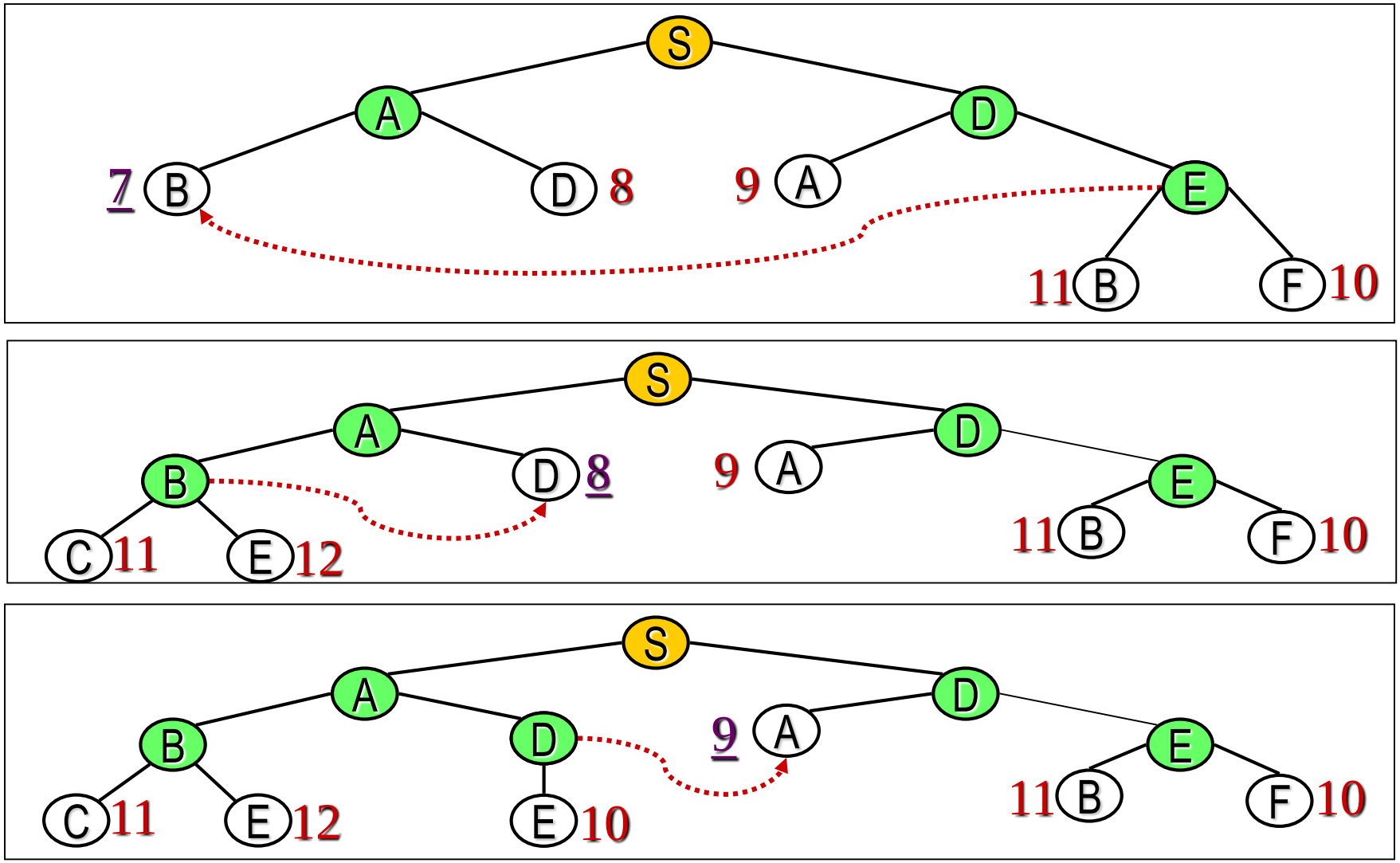


3.2 Búsqueda A*

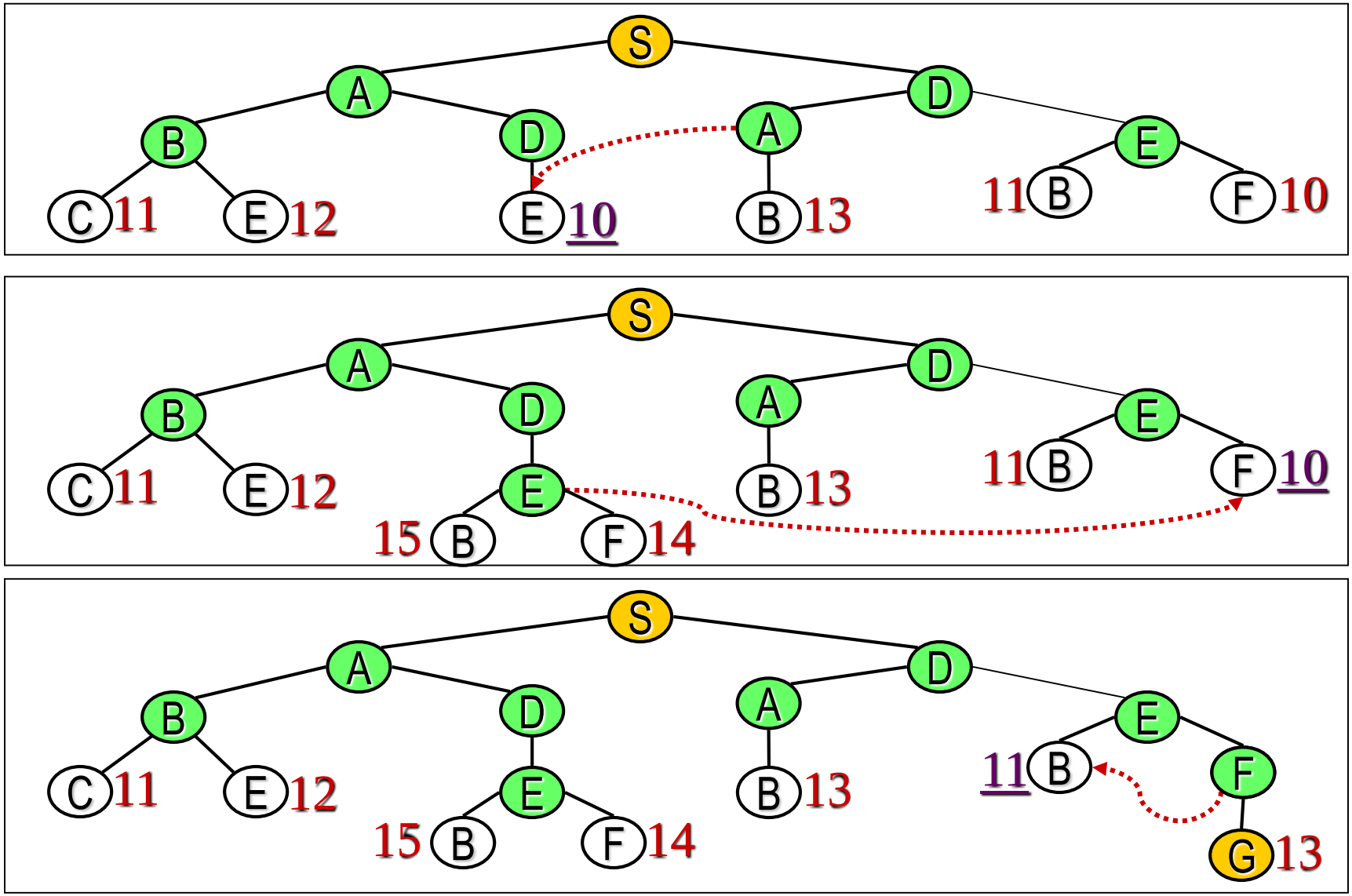
- Resolución por Búsqueda de Coste Uniforme



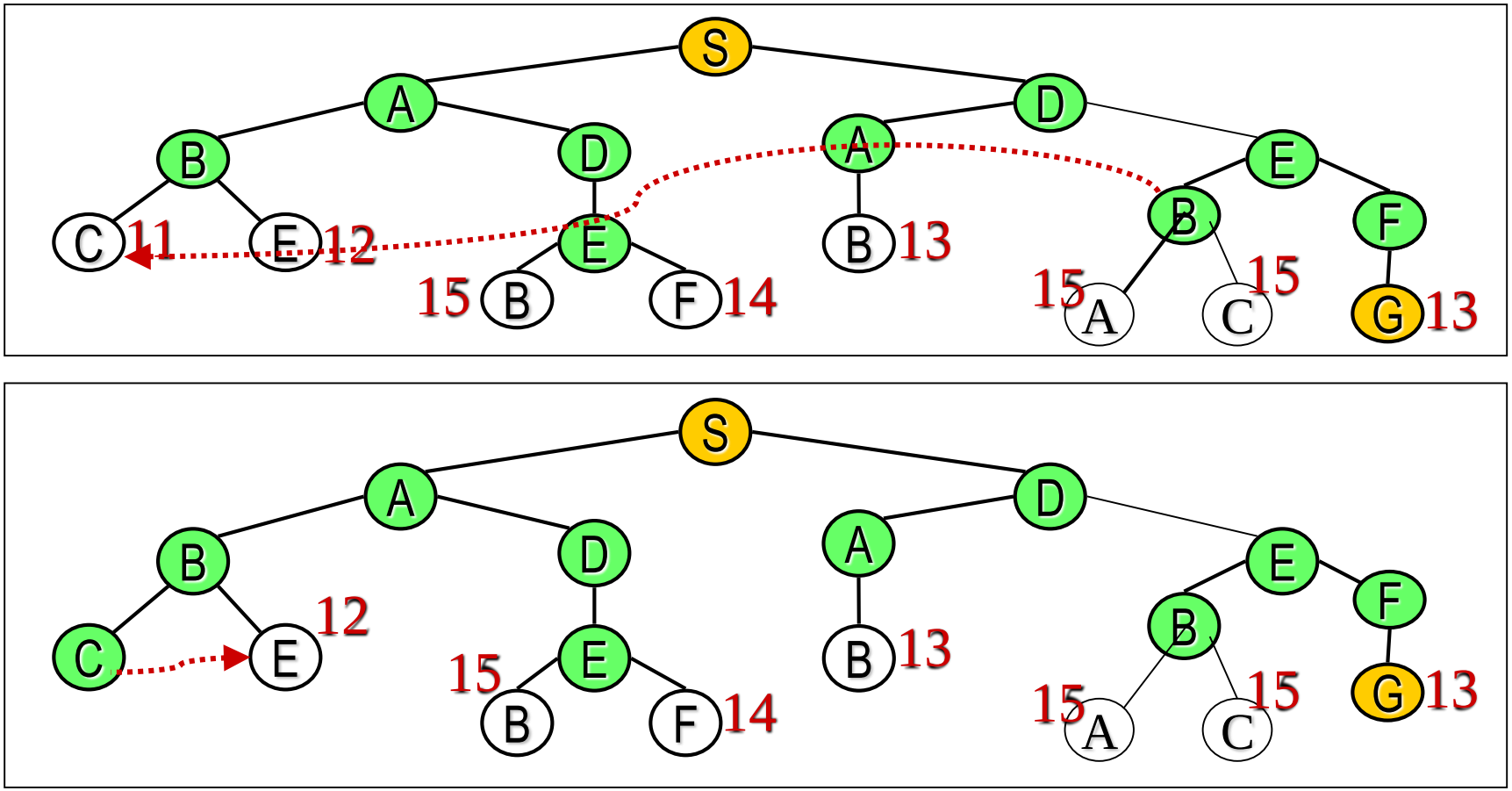
3.2 Búsqueda A*



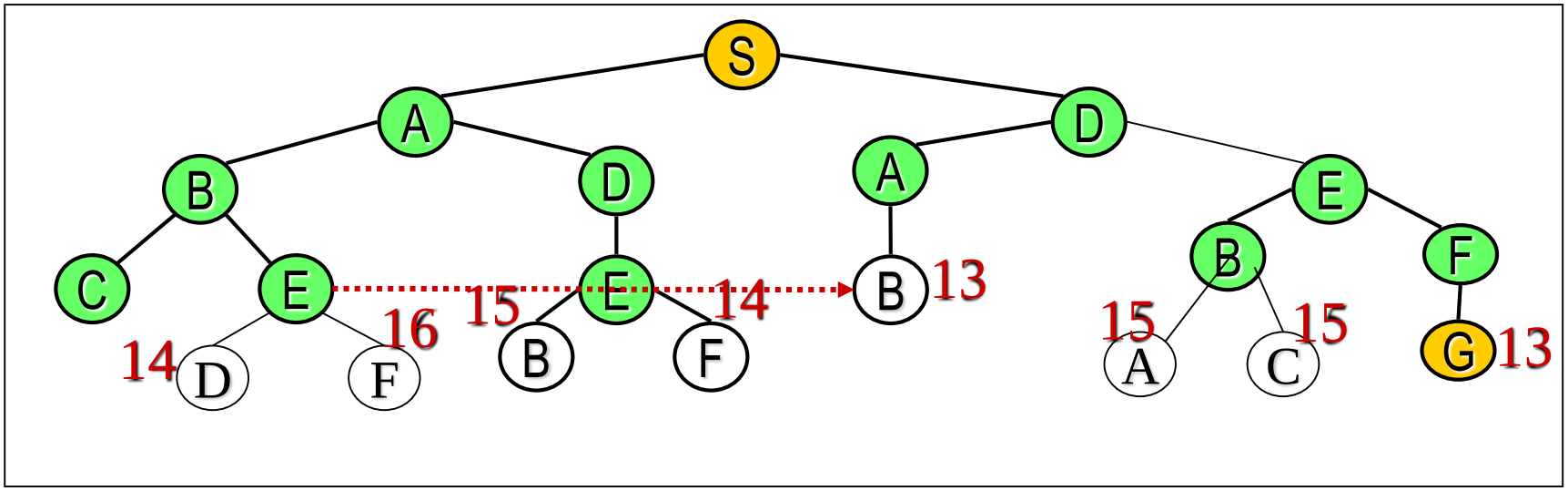
3.2 Búsqueda A*

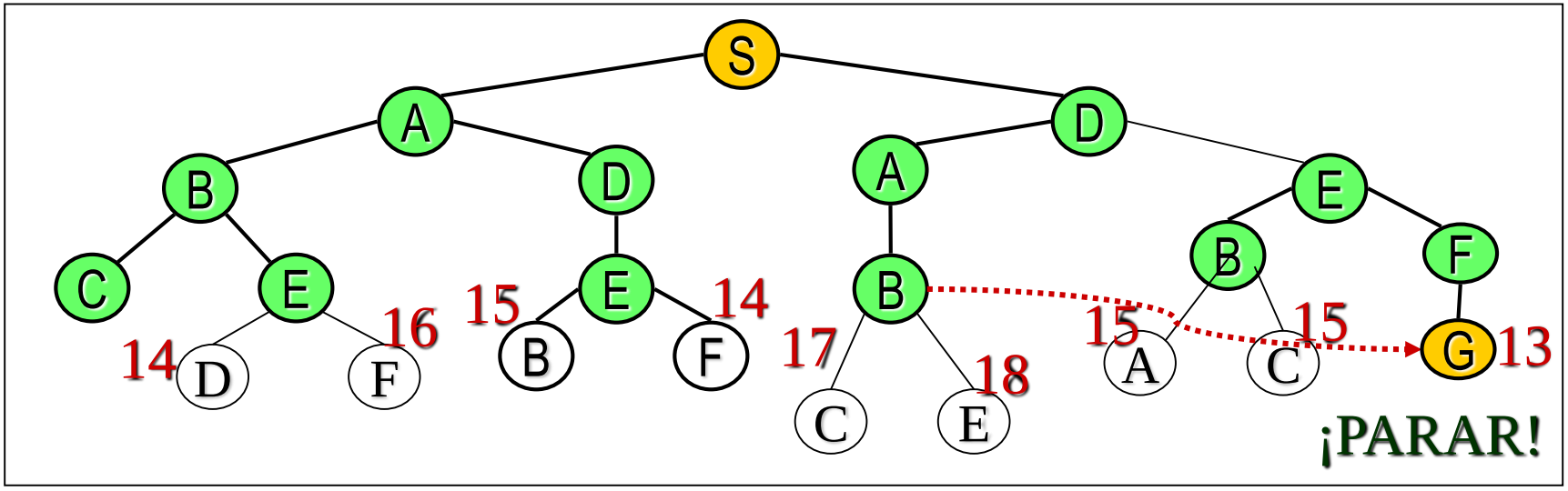


3.2 Búsqueda A*



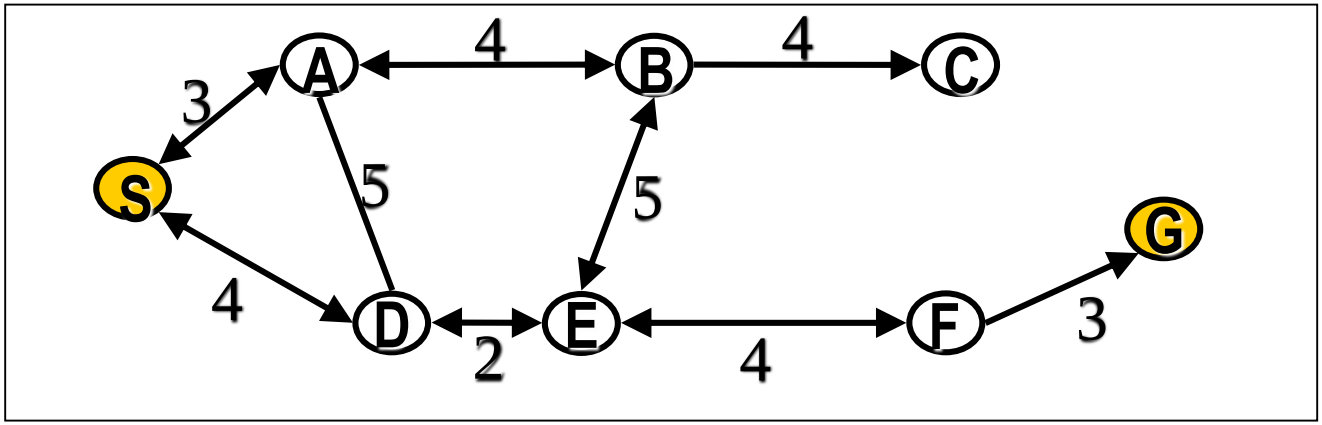
3.2 Búsqueda A*

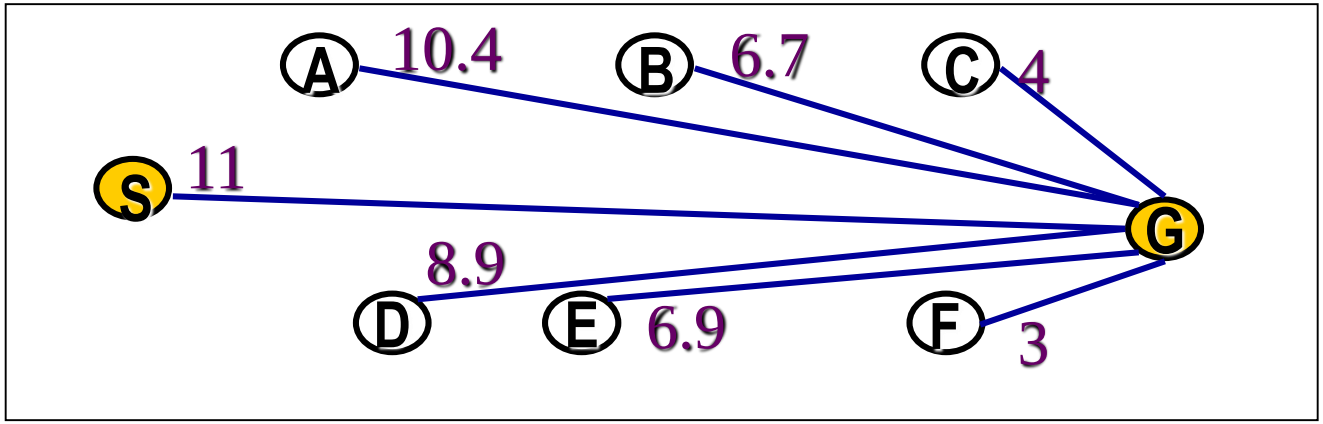




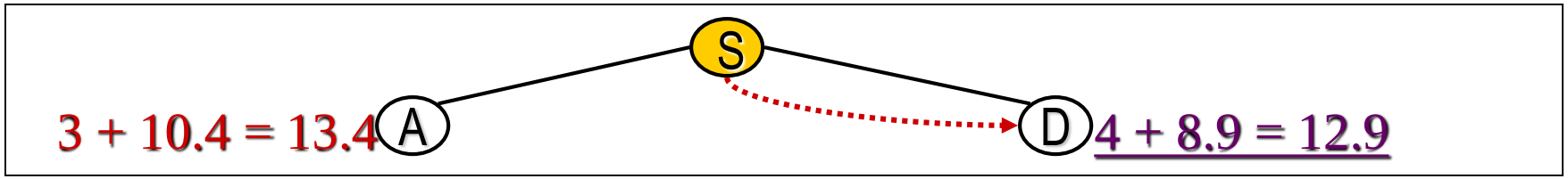
3.2 Búsqueda A*

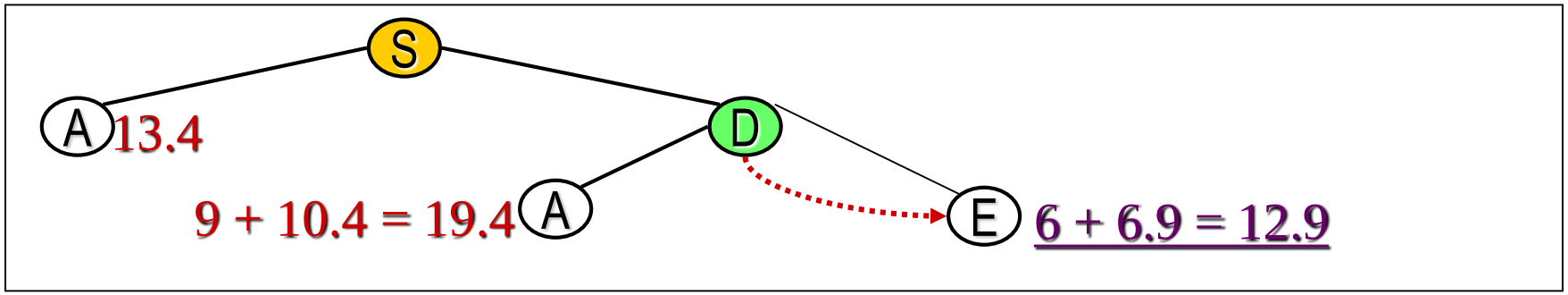
- Reconsiderar el problema incluyendo la heurística h_{DLR}

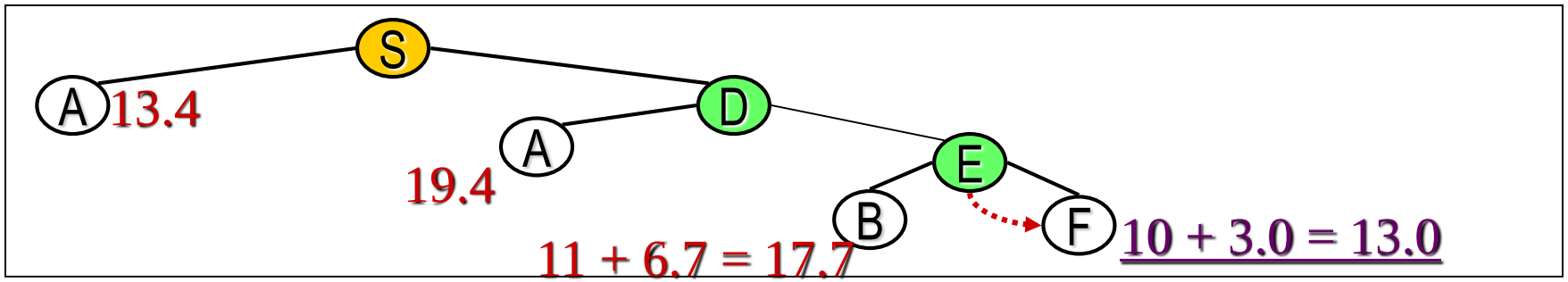




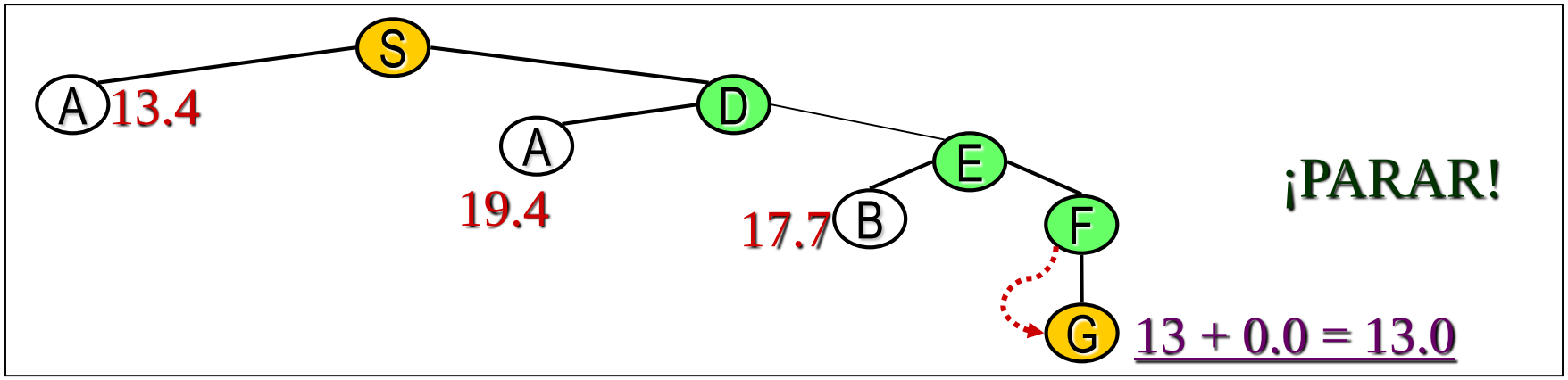
3.2 Búsqueda A*







3.2 Búsqueda A*



3.3 Variaciones de A*

- Algoritmos que mejoran A* acotando la memoria:
  - **IDA*:** A* con Profundidad Iterativa (*Iterative Deepening A**)
  - **MA*:** A* con Memoria acotada (*Memory Bounded A**)
  - **SMA*:** A*M Simplificada (*Simplified Memory Bounded A**)
    - Si al generar un sucesor falta memoria, se libera el espacio de los nodos de *abiertos* menos prometedores
  - **RTA*:** A* en Tiempo Real (Korf, 1988)

3.3 Variaciones de A*

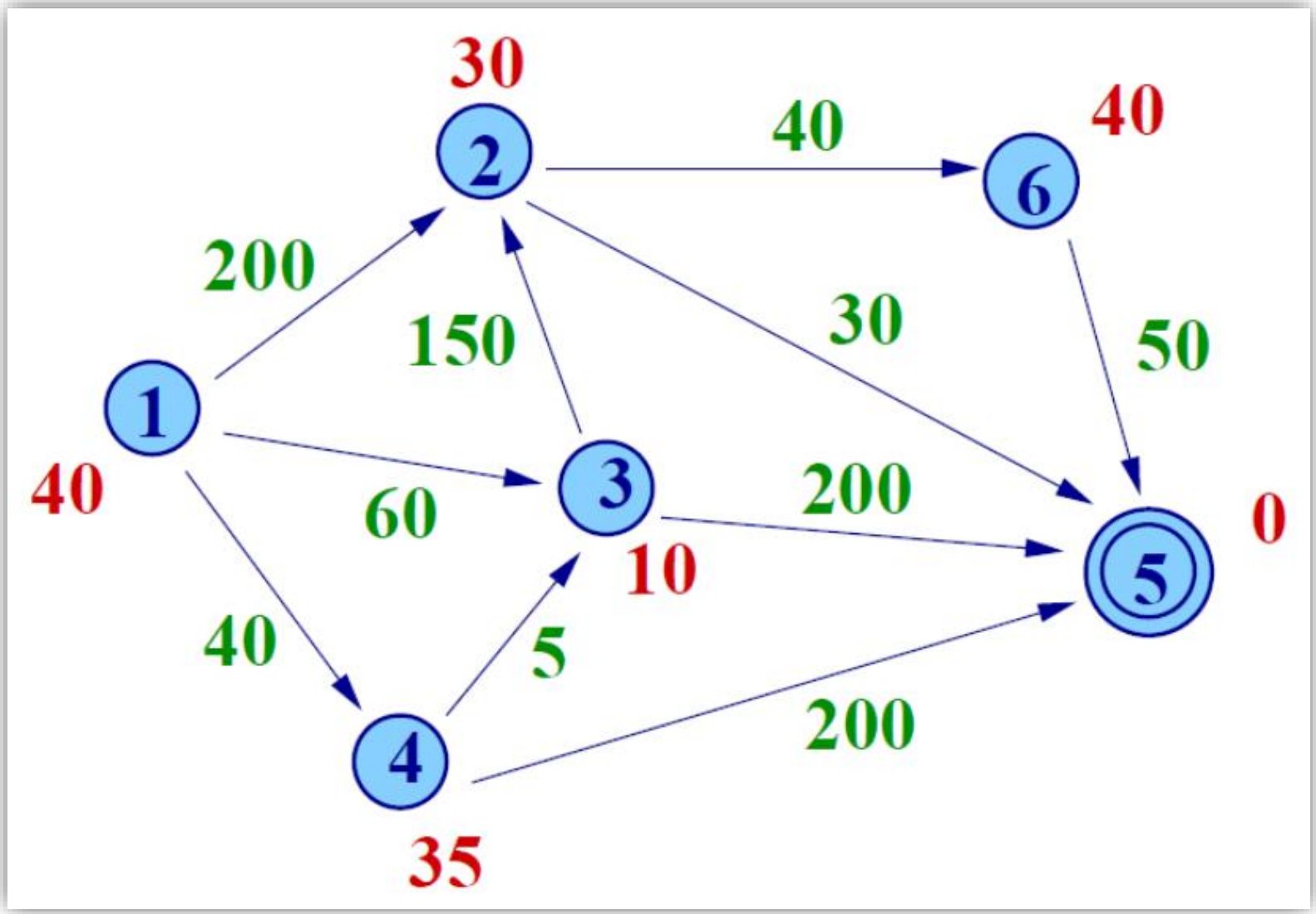
IDA*: A* con Profundidad Iterativa (Korf, 1985)

- Similar a la búsqueda ciega en Profundidad Iterativa
- Expande todos los nodos cuyo coste *f(n)* no excede un determinado valor
- η (*coste de corte*): valor mínimo de la función de coste en todos los nodos visitados pero NO expandidos
  - Iteración 1: $\eta_1 = h_0(\textit{nodo inicial})$
    - Expandir nodos según A* hasta que *f(nodos_sucesores)*> η
  - Si no es META,
    - Nueva iteración
    - Nuevo η sobre el conjunto de nodos todavía no expandidos

$$\eta = \min_{i=1,n}\{f(i)\} = \min_{i=1,n}\{g(i) + h(i)\}$$

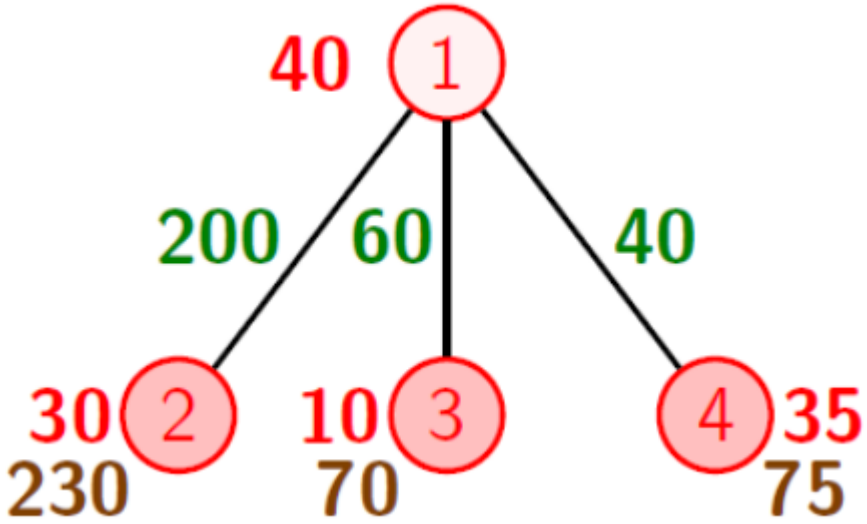
  - Repetir iteración

3.3 Variaciones de A*



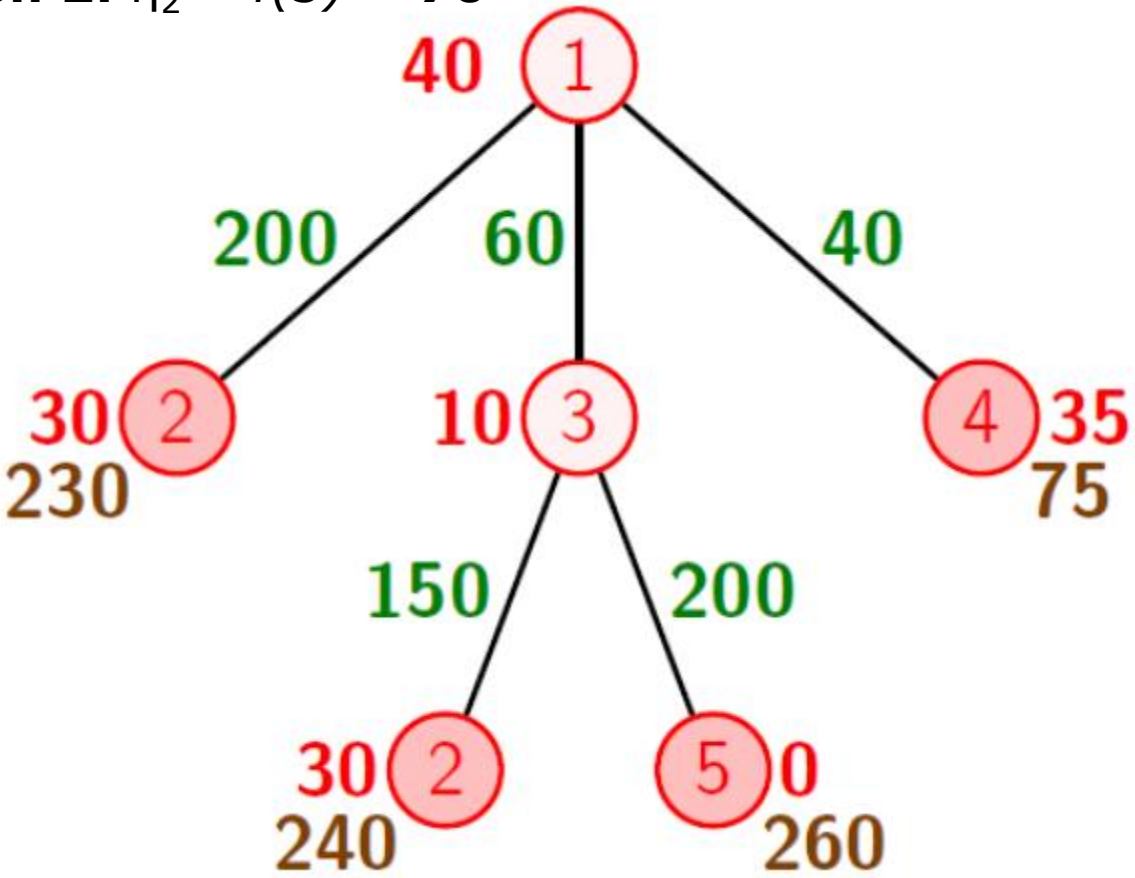
3.3 Variaciones de A*

Iteración 1: $\eta_1 = h(1) =$ **40**

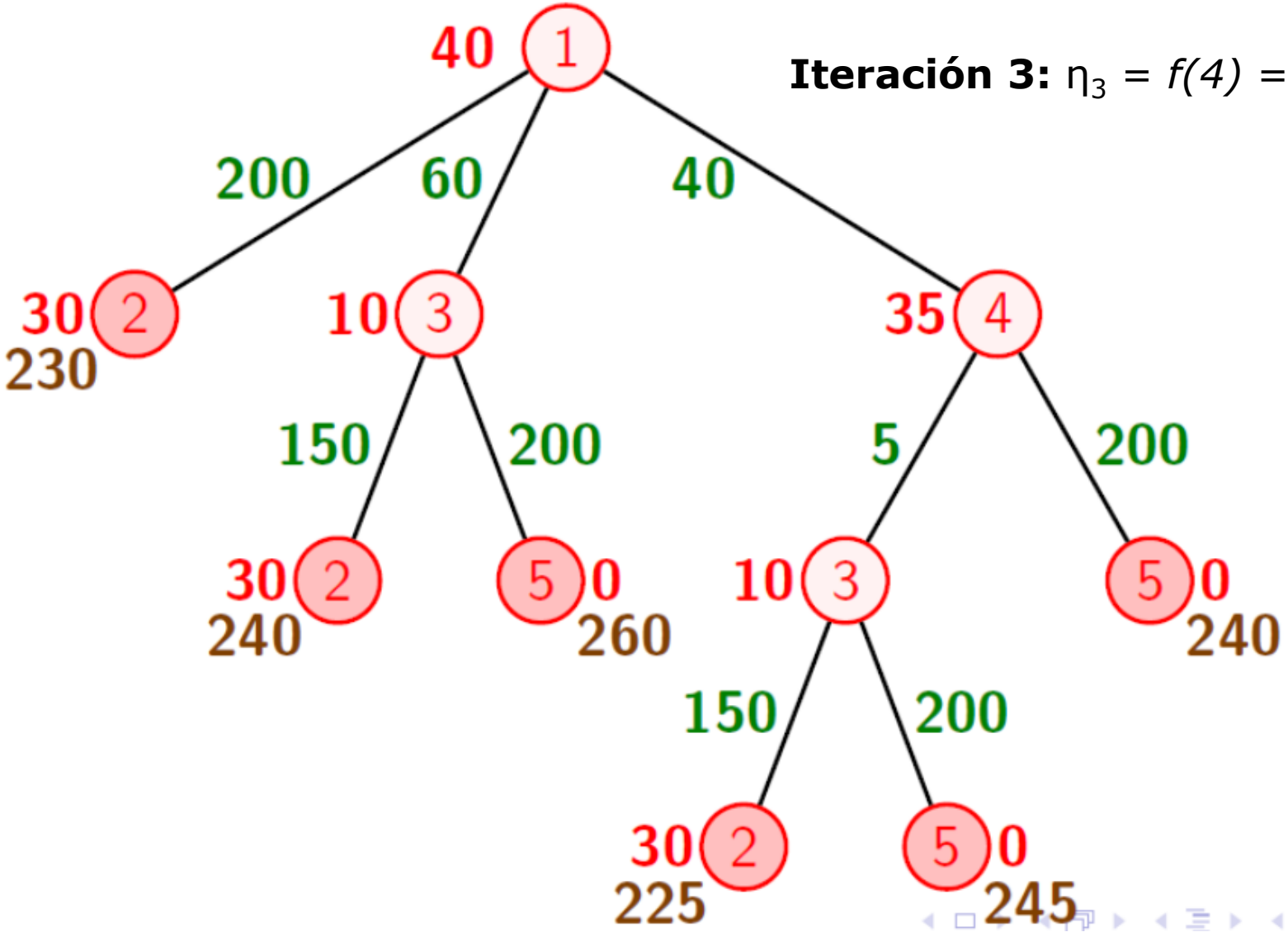


3.3 Variaciones de A*

Iteración 2: $\eta_2 = f(3) = \mathbf{70}$

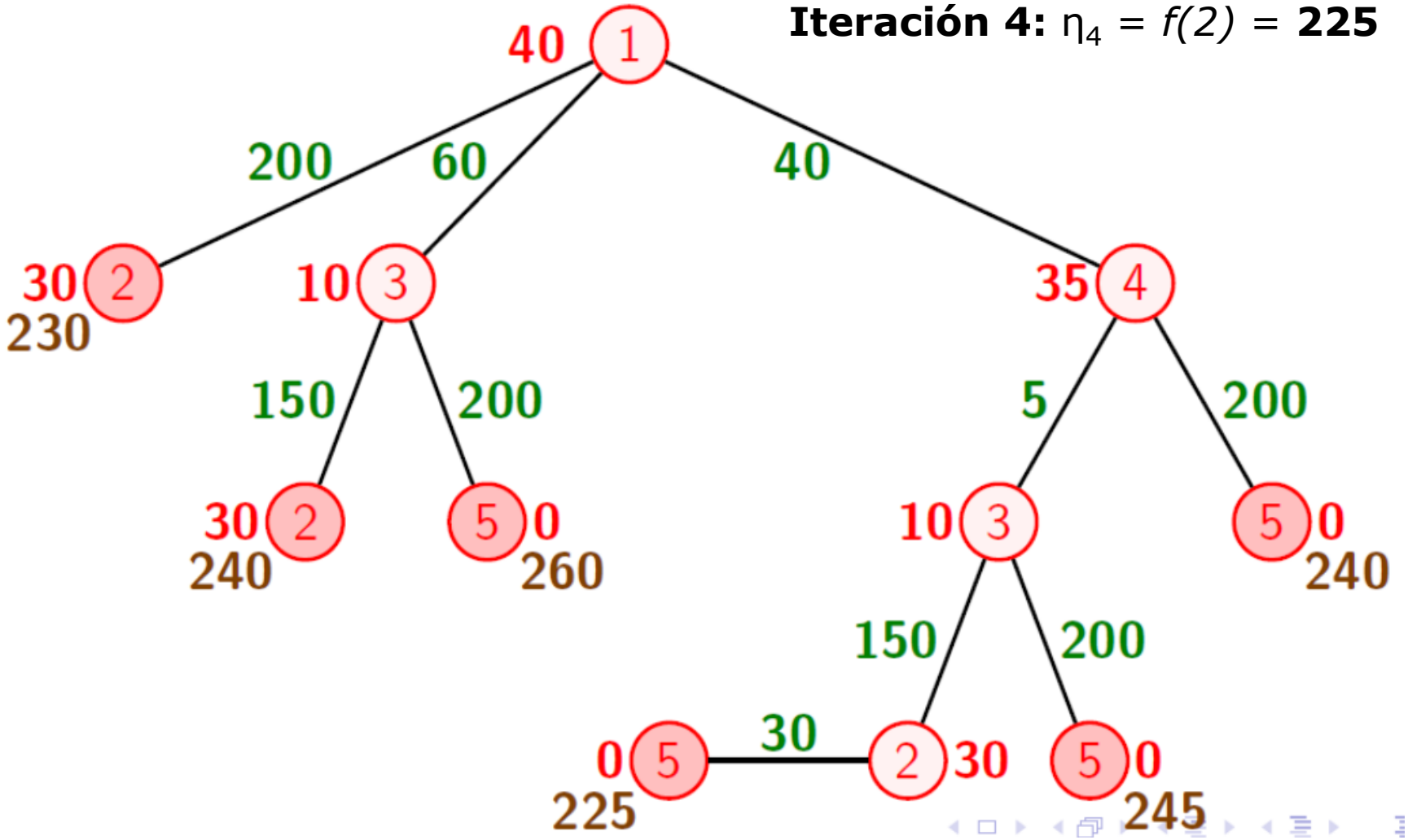


3.3 Variaciones de A*



Iteración 3: $\eta_3 = f(4) =$ **75**

3.3 Variaciones de A*



3.3 Variaciones de A*

- Propiedades
  - Completitud: es completo (encuentra solución si existe)
  - Complejidad tiempo: $O(b^d)$ Exponencial
  - Complejidad espacio: $O(b*d)$ lineal en la profundidad del árbol de búsqueda
  - Optima: **Si.** Es admisible y por lo tanto, encuentra la solución óptima
    - Aunque pudiera parecer lo contrario, el numero de re-expansiones es solo mayor en un pequeño factor que el numero de expansiones de los algoritmos PEM

- Fue el primer algoritmo que resolvió óptimamente 100 casos generados aleatoriamente en el 15–puzzle

1. Introducción
2. Funciones heurísticas
3. Búsquedas "primero el mejor"
    1. Búsqueda avara
    2. Búsqueda A*
    3. Variaciones de A*
4. Búsquedas iterativas
    1. Hill Climbing
    2. Simulated Annealing

4. Búsquedas iterativas

- Se usan cuando solo interesa el objetivo final
  - NO importa el camino ni su coste (¡¡ni se calcula!!)
  - 8-reinas, planificación, rutas…
- Reemplazan a las técnicas de búsqueda exhaustiva de forma eficiente
  - Comienzan con la configuración completa y hacen modificaciones para mejorar la calidad de la solución
  - No suelen almacenar caminos y buscan desde el estado actual hacia vecinos ➔ *algoritmos de búsqueda local*
  - *Lo típico es que no encuentran la mejor solución, pero pueden encontrar una solución aceptable.*

- Ventajas
  - usan poca memoria
  - funcionan en problemas continuos inmensamente grandes

4. Búsquedas iterativas

- Uso en problemas de optimización

  Búsqueda de los valores óptimos para los parámetros de un sistema que minimicen la función de coste

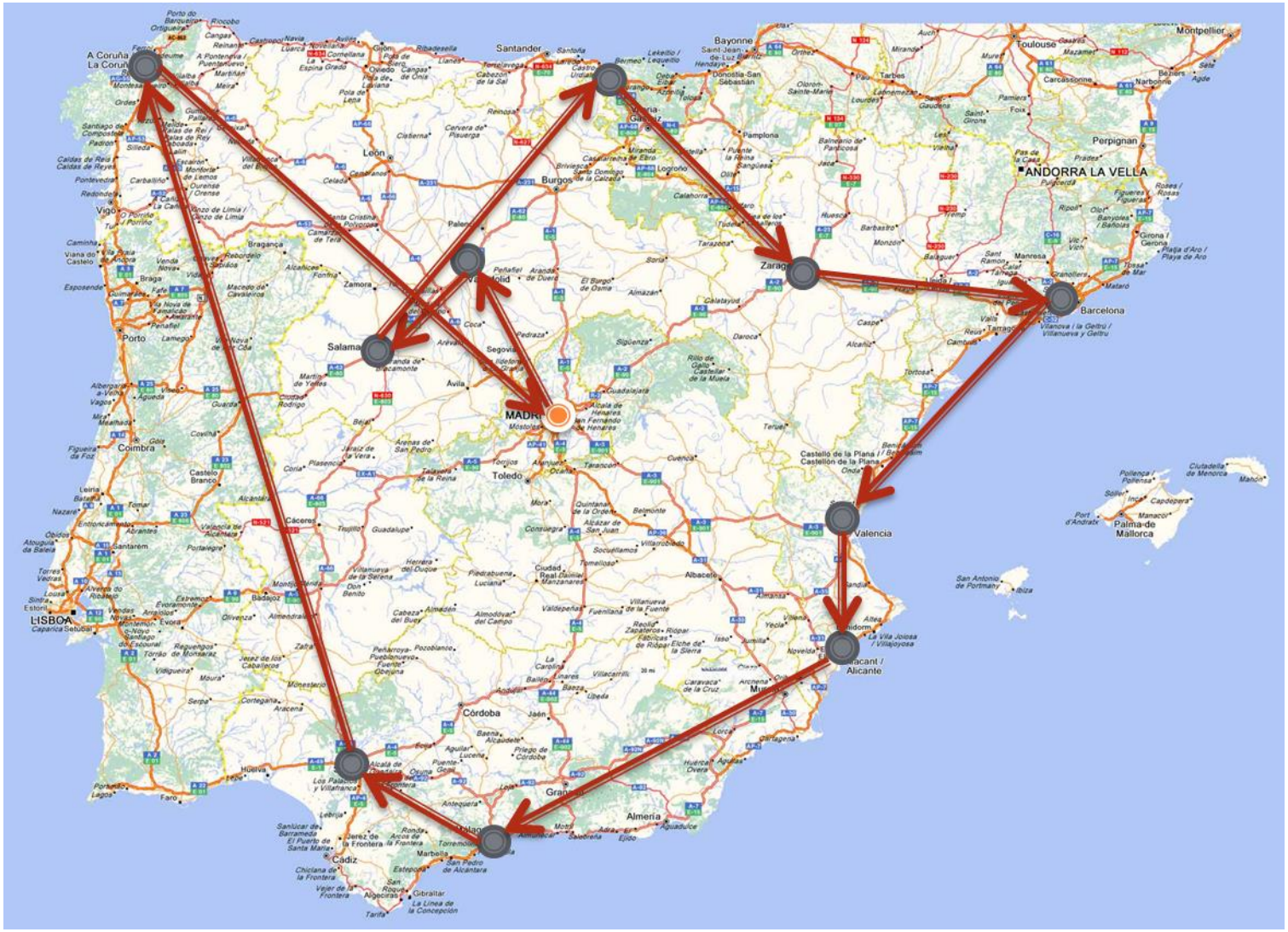
  - Buscan valles o picos en el paisaje formado por la función de coste

- Ejemplo: TSP
  - Un comerciante debe recorrer *N* ciudades, sin repetir ninguna, y volver a la ciudad de partida, en la mínima distancia.
  - El número de posibles rutas viene dado por permutaciones sin repetición (*N*!)
  - Para 2 ciudades (A, B), podemos hacer 2 recorridos:

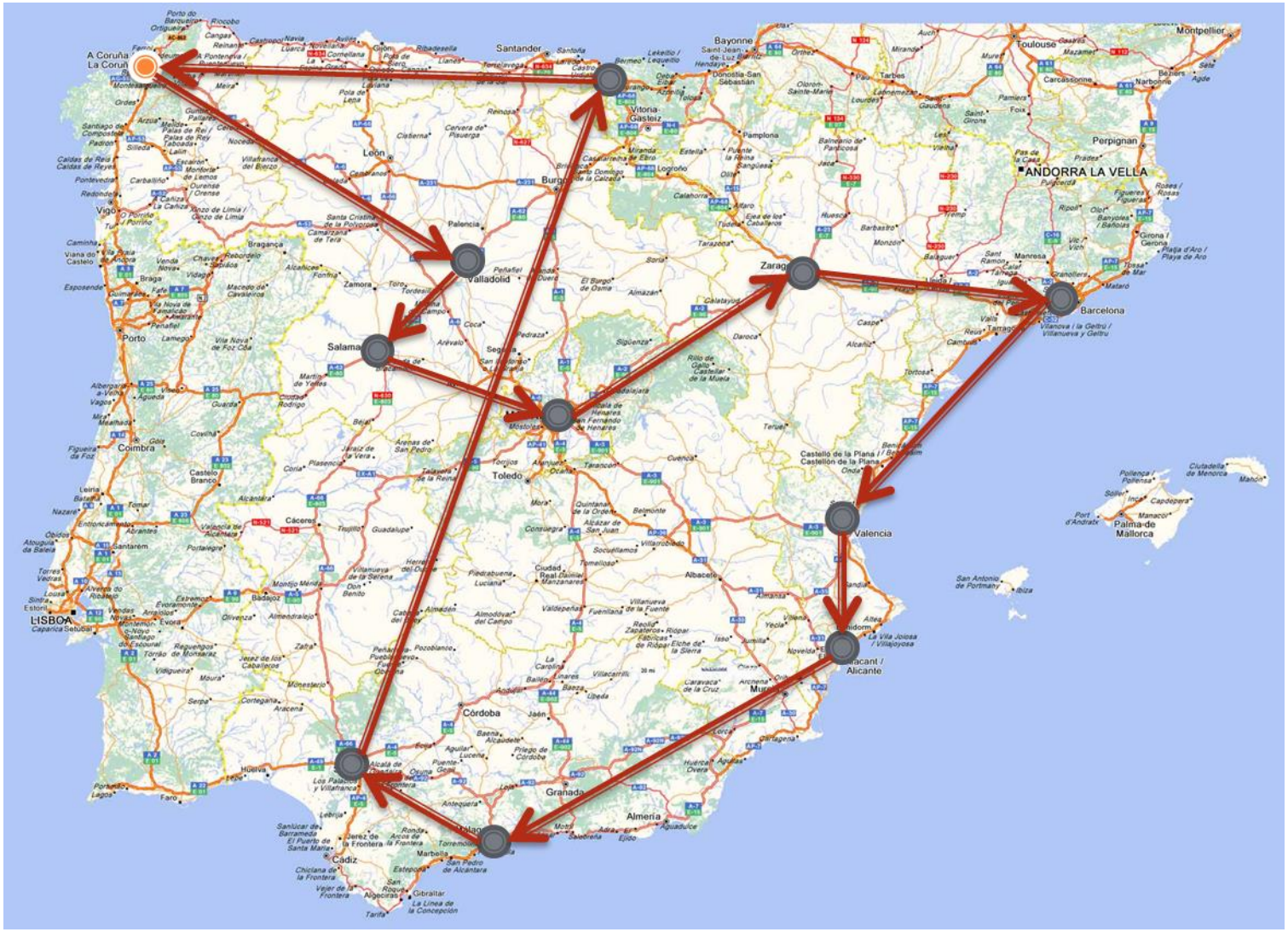
    (1) A → B → A

    (2) B → A → B
  - El resultado depende de la ciudad de partida

4. Búsquedas iterativas



4. Búsquedas iterativas

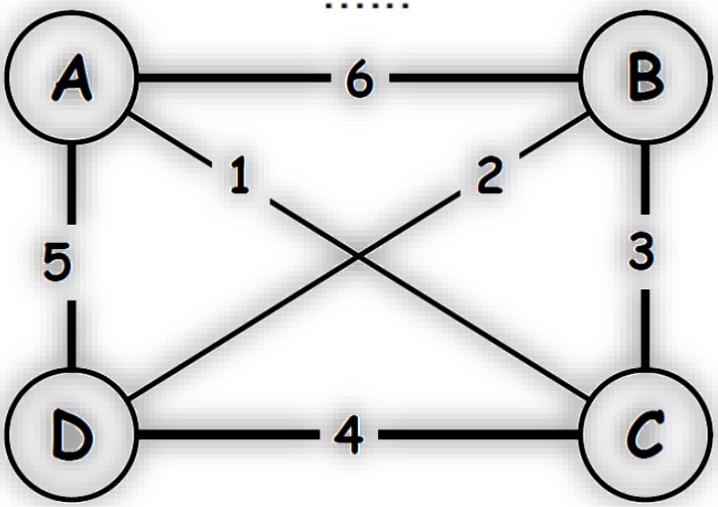


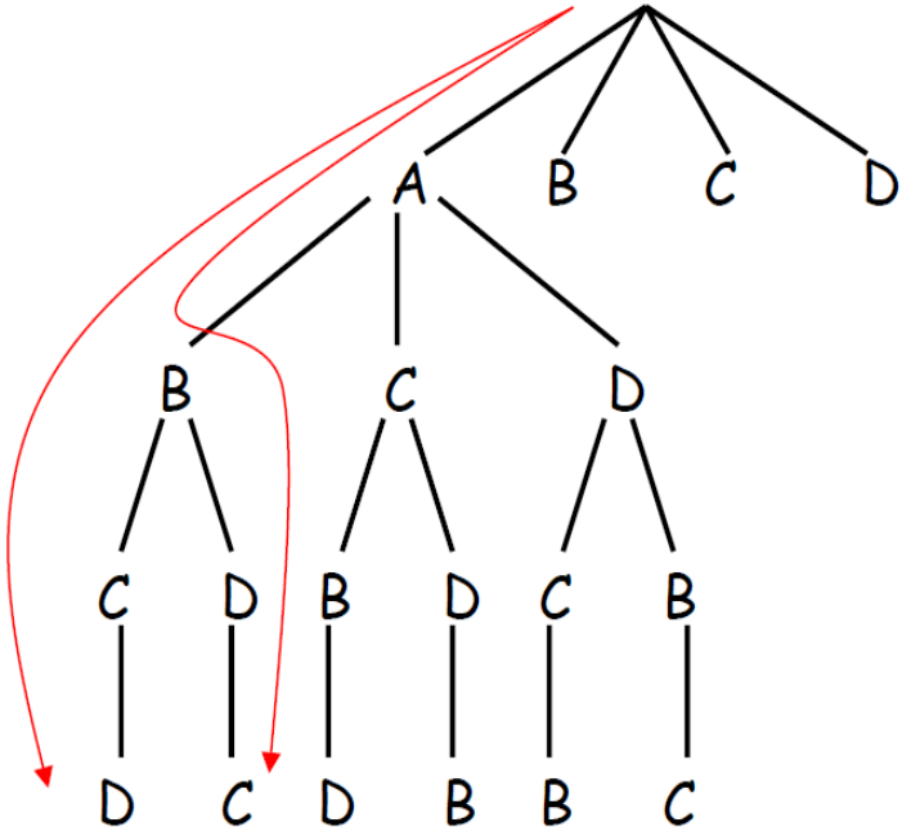
4. Búsquedas iterativas

- La generación de las posibles soluciones se lleva a cabo por orden alfabético de ciudades

1. A - B - C - D
2. A - B - D - C
3. A - C - B - D
4. A - C - D – B

.....





4. Búsquedas iterativas

- Otro ejemplo: Problema de las 8 reinas

4. Búsquedas iterativas

- Tipos de búsquedas iterativas

  - Optimización evolutiva
    - Algoritmos genéticos
  - Redes neuronales
    - Redes de Hopfield
  - Ya hemos visto estas búsquedas NO HEURISTICAS al ver la IA SUBSIMBÓLICA

  - Ascenso de gradiente (Hill climbing)

  - Simulated Annealing
    - Métodos de MonteCarlo
    - Máquina de Boltzmann

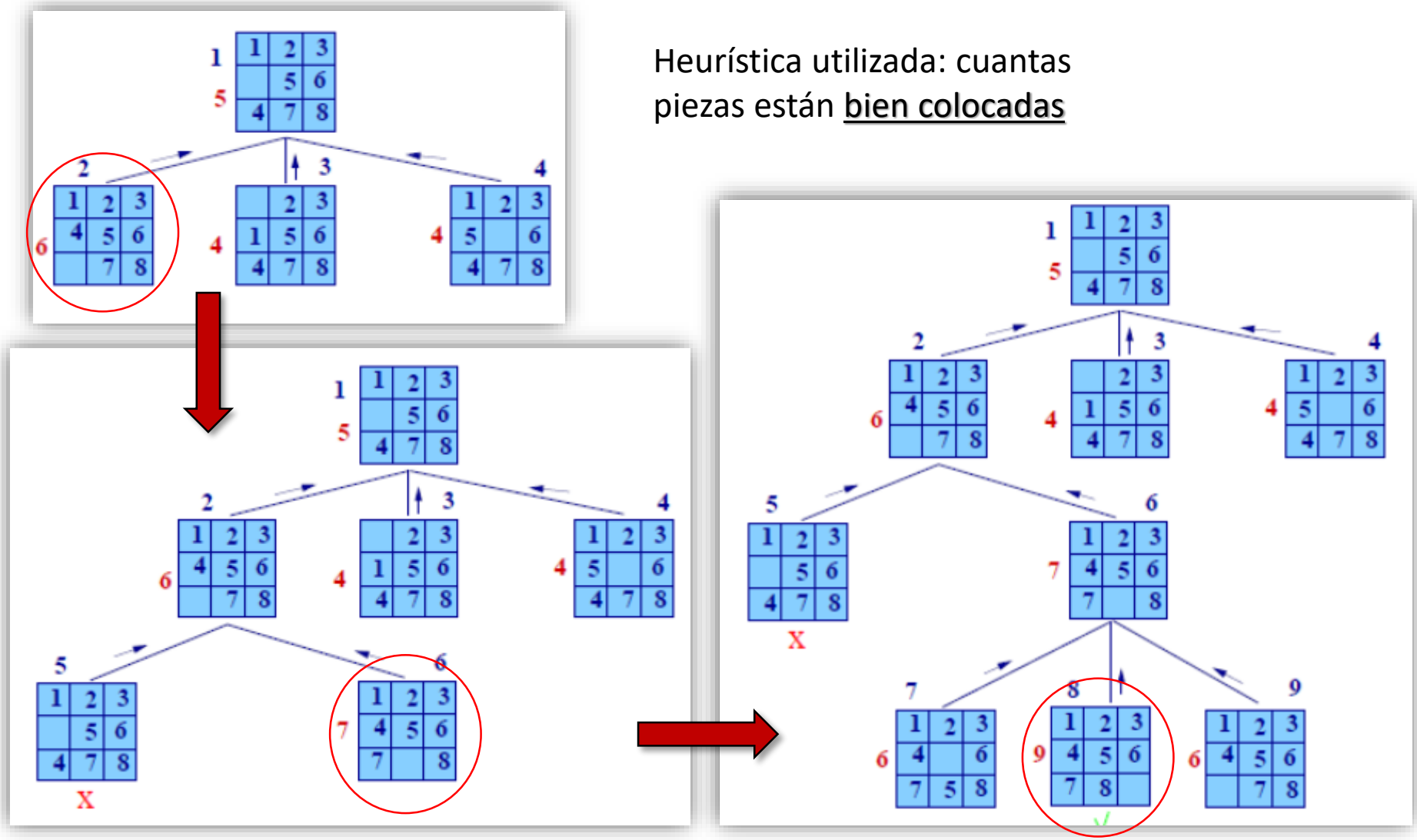
4.1 Hill Climbing

- Se llaman de escalada (o de ascensión a la colina) porque tratan de elegir en cada paso un estado cuyo ***valor heurístico sea mejor*** que el del estado activo en ese momento
  - Avanza al sucesor que *maximice la función de evaluación*
  - "muévete siempre a un estado mejor si es posible"

- Búsqueda "avara" local (no respecto al estado final)

- Suele funcionar bien, pero tiene problemas con la terminación
  - El proceso termina cuando en un estado dado, al aplicar todos los operadores ninguno de los estados resultantes es mejor
  - Puede atascarse en **máximos locales**, según el estado inicial

4.1 Hill Climbing



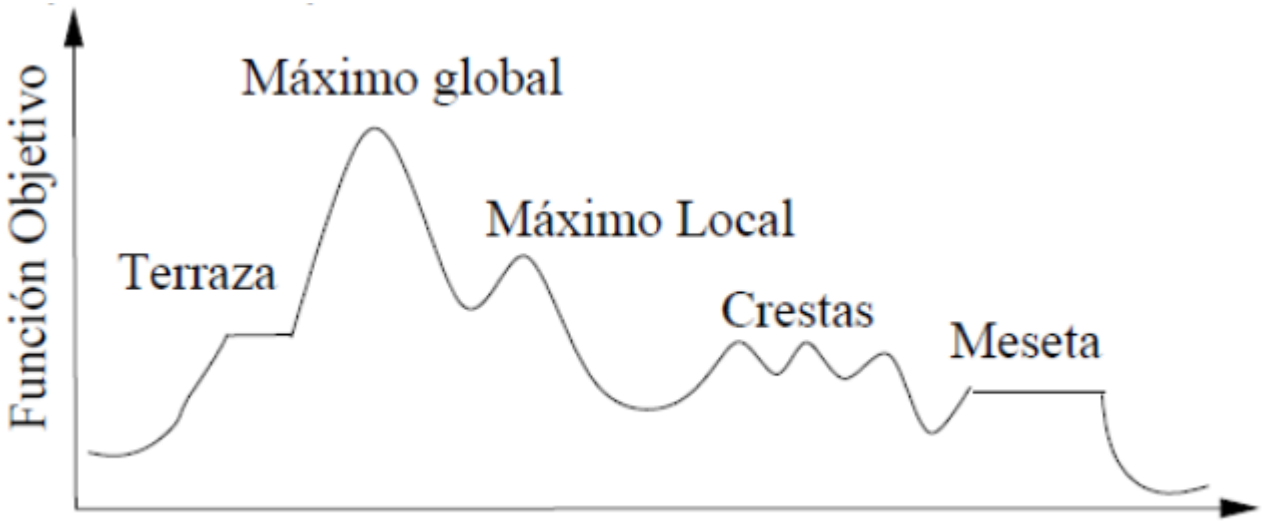
Heurística utilizada: cuantas piezas están bien colocadas

4.1 Hill Climbing

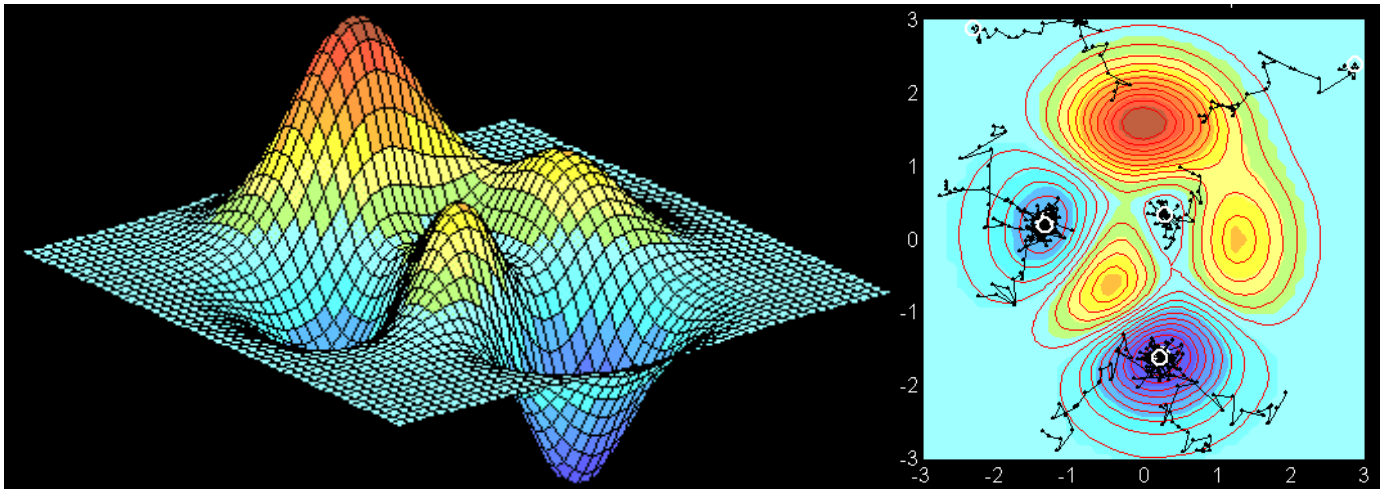
- Propiedades
  - Completitud: no tiene porque encontrar la solución
  - Admisibilidad: no siendo completo, aún menos será admisible
  - Eficiencia: rápido y útil si la función es monótona (de)creciente

- Problemas
  - Máximos (o mínimos) locales: pico que es más alto que cada uno de sus estados vecinos, pero más bajo que el máximo global
  - Mesetas: zona del espacio de estados con función de evaluación plana
  - Crestas: zona del espacio de estados con varios máximos (mínimos) locales

4.1 Hill Climbing

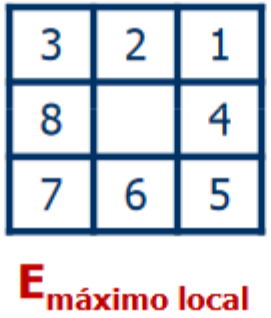


Espacio de Estados unidimensional



Espacio de Estados tridimensional

4.1 Hill Climbing



Máximo local

Todos los movimientos empeoran el valor de la función heurística.

Meseta

Todos los movimientos dejan igual el valor de la función heurística.

- Variantes (mejoran el problema del óptimo local)
  - Movimiento aleatorio entre ascendentes ➔ retroceso
  - Volver a comenzar la búsqueda ➔ reinicio aleatorio
  - Empeoramiento de la solución actual

4.2 Simulated Annealing

- Un modo de evitar que la búsqueda local finalice en óptimos locales es permitir que algunos movimientos sean hacia soluciones peores

- Pero si la búsqueda está avanzando realmente hacia una buena solución, estos movimientos de escape deben realizarse de un modo controlado

- Técnicas
  - Algoritmos Genéticos
  - Simulated Annealing
    - Desarrollado en 1993 para modelado de procesos físicos
    - Basado en el proceso metalúrgico de recalentamiento o "templado" (como las espadas)

4.2 Simulated Annealing

Algoritmo de búsqueda con un criterio probabilístico de aceptación de soluciones basado en Termodinámica

- Combina ascensión de colinas con movimientos aleatorios
- Busca unir eficacia y completitud para alcanzar el óptimo absoluto
  - La idea es escapar de los máximos locales permitiendo movimientos "incorrectos" (saltos hacia soluciones peores)
  - Controlando la frecuencia de estos movimientos mediante una *función de probabilidad* que reducirá gradualmente el tamaño y frecuencia de los saltos conforme avanza la búsqueda (y por tanto estamos más cerca, previsiblemente, del óptimo)
  - Acabando en un hill climbing normal.

4.2 Simulated Annealing

- En cada iteración se genera un número de vecinos

- *Probabilidad de salto*: Depende de la diferencia de coste entre la solución actual y la vecina (*ΔE*) y la Temperatura (*T*), según una distribución de Boltzmann

$$h(\Delta E, T) = \frac{1}{1 + e^{\frac{\Delta E}{cT}}}$$

  - Si *Δ*E < 0 (la solución vecina es mejor que la actual) se acepta el movimiento ya que conduce a un estado de menos energía
  - Si *Δ*E >0 (la solución vecina es peor) se puede aceptar el movimiento con más probabilidad cuanto mayor sea T
    - "calienta" ➔ salto. Aumenta la probabilidad de aceptar una solución peor
    - "enfría" ➔ ascensión. Disminuye la probabilidad de aceptar una solución peor

4.2 Simulated Annealing