Búsqueda Heurística III

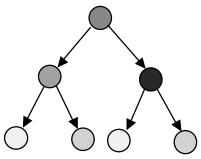
Pedro Meseguer
IIIA-CSIC
Bellaterra, Spain
pedro@iiia.csic.es

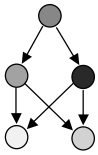
Búsqueda heurística

- Búsqueda informada por la función heurística $f(n)$
- Algoritmos: esquema primero el mejor (*best-first*)
  - Coste uniforme $f(n) = g(n)$
  - Búsqueda heurística pura $f(n) = h(n)$
  - A* $f(n) = g(n) + h(n)$
  - A* con heurísticas admisibles
- Evaluación:
  - calidad solución: ¿óptimo?
  - coste en tiempo: proporcional a los nodos generados
  - coste en memoria: proporcional a los nodos almacenados

Árboles y grafos

un solo camino para cada nodo





varios caminos para cada nodo

BÚSQUEDA EN ÁRBOL

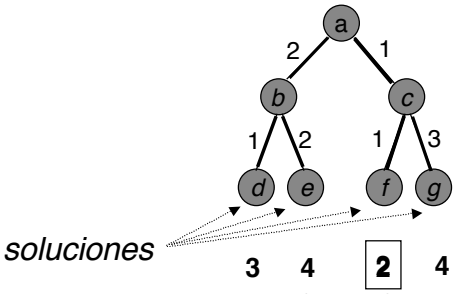
- Algoritmo más simple
- No detectan repetidos: si aparece un nodo ya expandido, se expande de nuevo (tiempo)

BÚSQUEDA EN GRAFO

- Algoritmo más complejo
- Almacenan nodos ya expandidos, para detectar repetidos (gastan memoria)

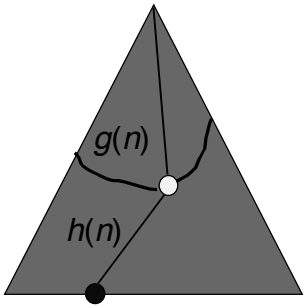
Coste en arcos

- Arcos:
  - en búsqueda ciega, suponemos cada arco cuesta 1
  - solución óptima: más cercana a la raíz
  - ahora arcos con costes arbitrarios (mínimo $e > 0$)
  - camino óptimo: mínimo coste a una solución



Funciones de coste

- $g(n)$: suma de costes desde la raíz hasta n
- $h(n)$: estimación de coste mínimo desde n hasta una solución



$f(n) = g(n)$

$f(n) = h(n)$

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

Esquema de primero el mejor (*best-first*) (búsqueda en árbol)

1. $L \leftarrow$ nodo raíz.
2. Si L vacío, fallo, stop.
   Sino, $n \leftarrow$ extrae -mínimo-f(L)
3. Si n es solución, retornar el camino desde la raíz, stop.
   Sino, generar los sucesores de n.
4. Añadir a L los sucesores de n,
   etiquetando sus respectivos caminos desde la raíz. Ir a 2.

Diferencias con búsqueda ciega

Siguiente nodo: en lugar de escoger el primero de L, se escoge el de mínimo coste

1. *L* ← nodo raíz.
2. Si *L* vacío, fallo, stop.
   Sino, *n* ← **extrae -mínimo-f(*L*)**
3. **Si *n* es solución**, retornar el camino desde la raíz, stop.
   Sino, generar los sucesores de *n*.
4. **Añadir** a *L* los sucesores de *n*,
   etiquetando sus respectivos caminos desde la raíz. Ir a 2.

No hay orden al añadir sucesores

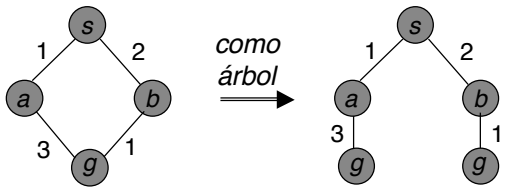
Test de solución: no se hace al generar el nodo sino al expandir el nodo

Test al expandir

f(n) = g(n)

s: inicio

g: objetivo



1. expandimos *s, L* = {*a*(1), *b*(2)}
2. expandimos *a*, *L* = {*b*(2), *g*(4)}

camino s-a-g, coste = 4

3. expandimos *b*, *L* = {*g*(3), *g*(4)}
4. expandimos *g*(3), solución

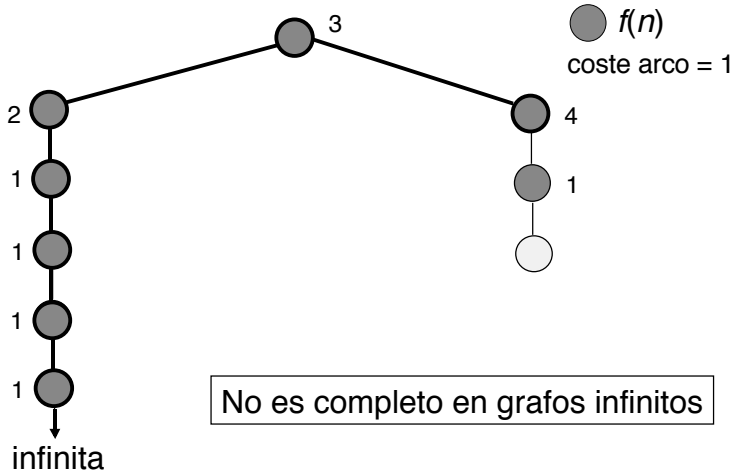
camino s-b-g, coste = 3

Coste uniforme $f(n) = g(n)$

- Calidad de la solución: encuentra camino de mínimo coste
- Tiempo:
  - c : coste de la solución
  - e : mínimo coste de un arco
  - antes de la solución, expandirá todos los nodos con coste $\leq c$ (caso peor)
  - complejidad $O(b^{c/e})$
- Espacio:
  - si todos los arcos cuestan lo mismo: degenera en *anchura*
  - complejidad $O(b^{c/e})$

$f(n) = h(n)$

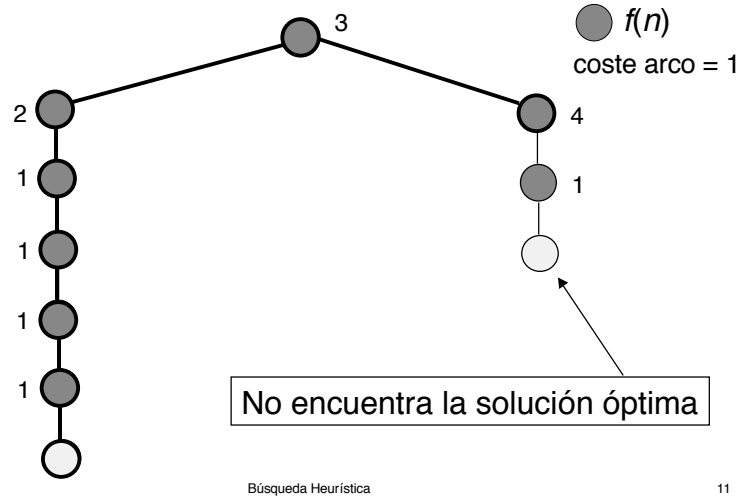
Búsqueda heurística pura (*pure heuristic search*)



No es completo en grafos infinitos

$f(n) = h(n)$

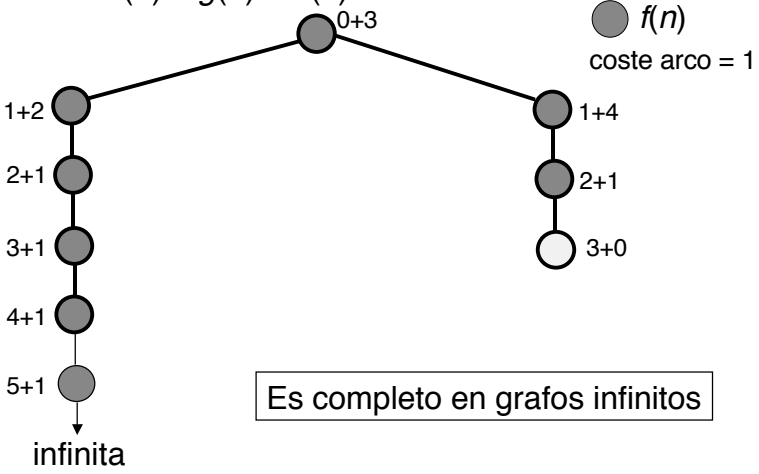
Búsqueda heurística pura (*pure heuristic search*)



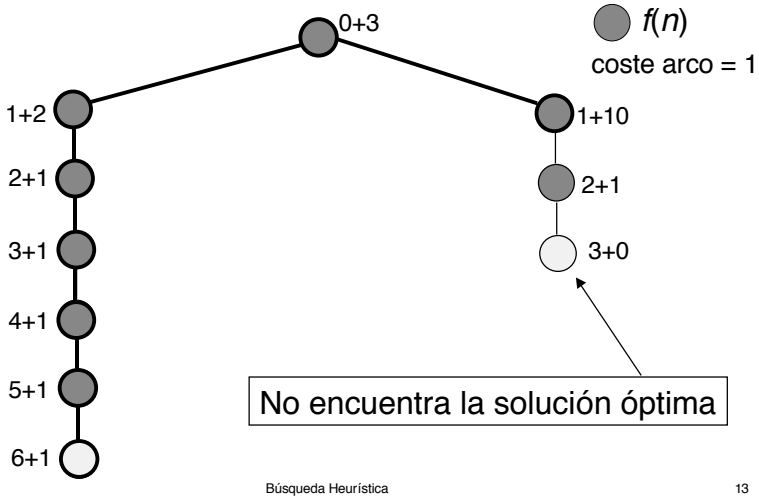
A*: $f(n) = g(n) + h(n)$

Problema: $h(n)$ no incluye el coste del camino ya recorrido

Solución: $f(n) = g(n) + h(n)$



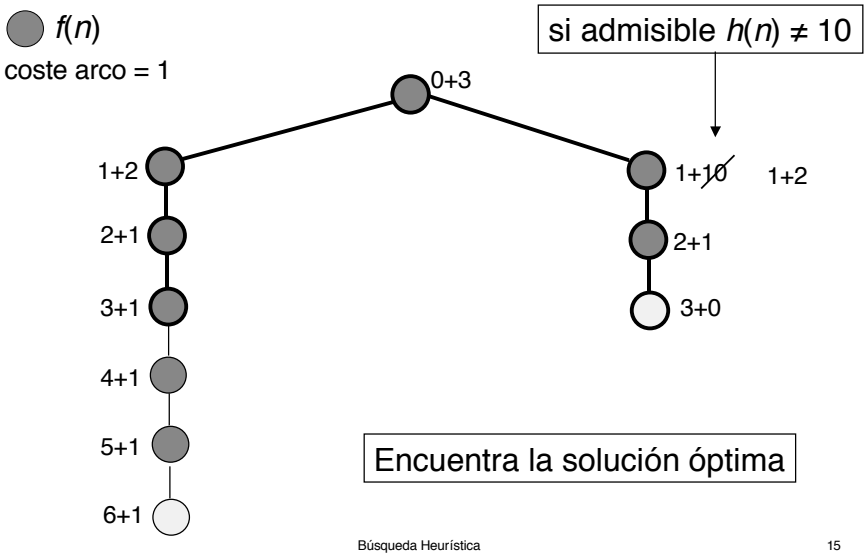
A*: $f(n) = g(n) + h(n)$



A*: *h* admisible

- $h(n)$: estimación de coste mínimo desde n hasta una solución
- $h^*(n)$: coste mínimo desde n hasta una solución
- h **admisible** ssi $h(n) \leq h^*(n) \quad \forall n$
- **Si *h* admisible, A* encuentra la solución óptima**

A*: *h* admisible

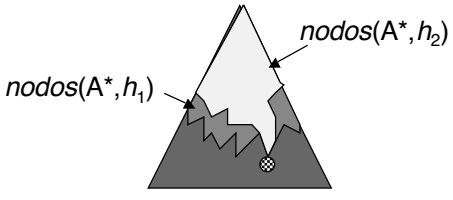


A* heurísticas más informadas

- *nodos* (A*, *h*): nodos expandidos por A* con *h*

- h_1 y h_2 admisibles,
- h_2 es más informada que h_1 sii $h_2(n) > h_1(n)$ $\forall n$

$$nodos\ (A^*, h_2) \subset nodos\ (A^*, h_1)$$

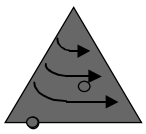


A* casos límite

- Heurística nula

$h(n) = 0 \quad \forall n$

- $f(n) = g(n)$
- A* degenera en búsqueda en anchura



espacio exponencial 

- Heurística perfecta

$h(n) = h^*(n) \quad \forall n$

- $f(n)$ = coste óptimo
- puede desarrollar todos los caminos con coste mínimo
- basta con un camino
- desempatar con g mayor

A* sobre grafos

- Árbol: un solo camino para cada nodo (*supuesto hasta ahora*)
- Grafo: varios caminos para cada nodo (*situación real*)
- Pregunta: ¿podemos adaptar A* de árboles a grafos?
  - La primera vez que se genera n, hay un único camino c_1, coste $g(c_1)$
  - Se puede volver a generar n por otro camino c_2, coste $g(c_2) \neq g(c_1)$
  - A* ¿cómo puede manejar dos caminos con distinto coste si
    - n aún no ha sido expandido?
    - n ya ha sido expandido?

A*: *OPEN* / *CLOSED*

- *OPEN*:
  - sustutuye a la lista *L*
  - contiene todos los nodos generados pero no expandidos
- *CLOSED*: contiene todos los nodos expandidos

A almacena todo el espacio generado*

- *n* generado por camino c_1,
  - se vuelve a generar *n* por camino c_2
  - hay que quedarse con camino de menor coste
  - si $n \in OPEN$, quedarse con camino min $\{g(c_1)\ \ g(c_2)\}$ (*sencillo*)
  - si $n \in CLOSED$, *n* ya ha sido expandido. Si $g(c_2) < g(c_1)$, hay que ir a los descendientes de *n* y actualizar su coste vía camino c_2 (*tostón*)

Consistencia y monotonía

- *h* consistente:

$$h(n) \le k(n,m) + h(m) \qquad \forall\ n, m,$$

$k(n,m)$: mínima distancia entre *n* y *m*

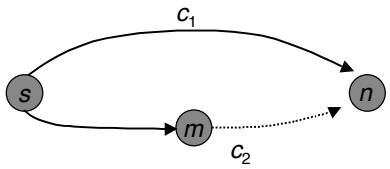
- *h* monótona:

$$h(n) \le c(n,n') + h(n') \qquad \forall\ n, n' \in succ(n)$$

- *h* monótona $\Rightarrow f(n') \ge f(n)$, no decrece en un camino
- consistencia $\Leftrightarrow$ monotonía
- consistencia / monotonía $\Rightarrow$ admisibilidad

Monotonía y $g(n)$

Si h monótona, $g(n) = g^*(n)$ (óptima) para nodos expandidos



- Sea $n \in$ *open* por c_1, seleccionado para expansión, con $g(n) > g^*(n)$
- Existe c_2 camino óptimo hasta n; sea m el último nodo generado de c_2
- El camino c_2 hasta m es óptimo: $g(m) = g^*(m)$. ¿Cómo es $f(m)$?

$f(m) = g^*(m) + h(m) \leq g^*(m) + k(m,n) + h(n) = g^*(n) + h(n) < g(n) + h(n) = f(n)$

consistencia

- Contradicción con que n sea el nodo con f mínima! Por lo tanto, $g(n) = g^*(n)$.

Monotonía y A*

- A* expande n: n pasa de *OPEN* a *CLOSED*
- h monótona:

  todo n expandido tiene $g(n) = g^*(n)$ (óptima)

  $\Leftrightarrow$

  todo $n \in$ *CLOSED* tiene $g(n) = g^*(n)$ (óptima)

- Por tanto: si $n \in$ *CLOSED*, su camino es el óptimo

A* sobre grafos no ha de propagar el coste de descubrir mejores caminos para los nodos ya expandidos.

A*: *path-max*

- ¿Es posible generar heurísticas monótonas?
- SÍ. Supongamos *h* admisible pero no monótona
- Redefinimos *f* como

$$f(n') = \max (f(n), g(n') + h(n')) \qquad n' \in succ(n)$$

- *f* es no decreciente, genera una nueva *h* monótona

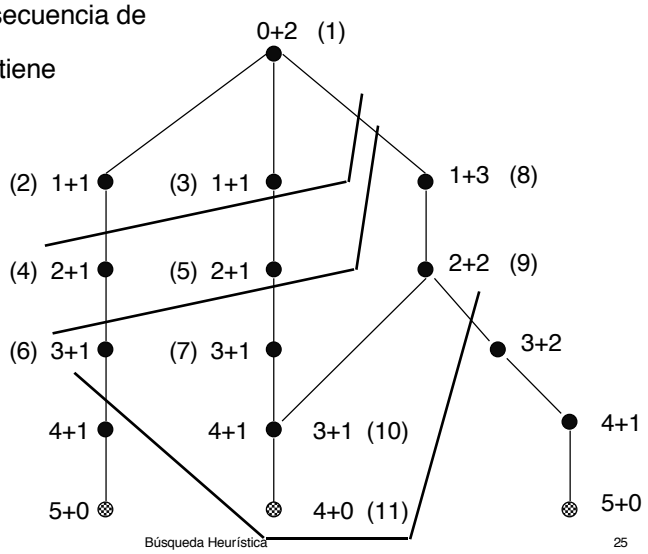
A* sobre grafos

1. *OPEN* ← nodo raíz
2. Si *OPEN* vacía, fallo, stop. Sino, *n* ← extrae-mínimo-f(*OPEN*)
3. Si *n* es objetivo, éxito, stop. Sino, añadir *n* a *CLOSED* y generar los sucesores de *n* (expansión de *n*).
4. Para cada sucesor *n'* hacer:
   - 4.1 Si *n'*∈*OPEN,* reetiquetar *n'* con el camino más corto desde la raíz.
   - 4.2 Si *n'*∈*CLOSED,* ignorar *n'.*
   - 4.3 Si *n'*∉*OPEN* y *n'*∉*CLOSED,* añadir *n'* a *OPEN*, etiquetándolo con su camino desde la raíz.

4.4 Ir al paso 2.

A*: contornos

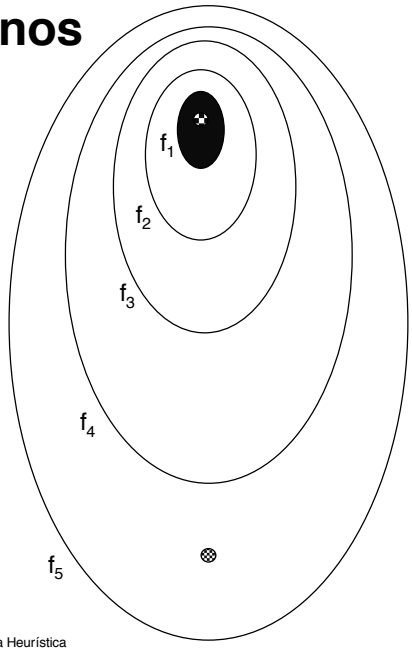
Si *h* monótona, la secuencia de nodos expandidos tiene *f* no decreciente



A*: contornos

Nodos estáticamente ordenados por f

Si *h* monótona, A* expande todos los nodos en el nivel f_i antes de expandir los de f_{i+1}



A* resumen

- A* = primero el mejor con $f(n) = g(n) + h(n)$
- Completo
- Si *h* admisible, A* encuentra solución óptima
- Heurísticas más informadas → menos nodos
- Si *h* no es admisible por ε, A* se pasa en solución óptima por ε
- No hay otro algoritmo que lo haga mejor con la misma información
- Degenera en consumo exponencial de memoria