UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Búsqueda A* [1]

Albert Sanchis
Alfons Juan

DSIC

Departamento de Sistemas
Informáticos y Computación

[1] Para una correcta visualización, se requiere Acrobat Reader v. 7.0 o superior

Objetivos formativos

- Aplicar el algoritmo A^*.
- Construir el árbol de búsqueda A^*.
- Analizar la optimalidad y complejidad de búsqueda A^*.
- Distinguir A^* estándar de la versión sin nodos cerrados.

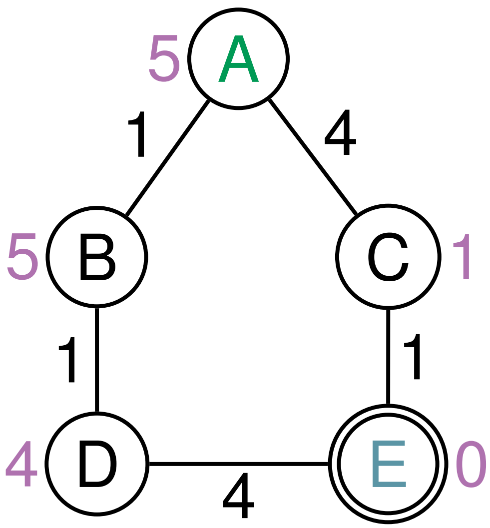
Índice

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1. Introducción

Búsqueda A** consiste en enumerar caminos hasta encontrar una solución, priorizando los de menor coste total estimado ($f = g + h$) y evitando ciclos:

Grafo de estados



Búsqueda

A 0+5

AB 1+5
AC 4+1

ACE 5+0 sol.!

A* generaliza UCS con la introducción de la ***heurística*** h de estimación del coste mínimo de llegar a meta; no negativa, nula en meta.

2. El algoritmo A* [1]

A* (G, s', h) // G grafo ponderado, s' *start*, h ***heurística***

  $O = IniCola(s', f_{s'} \triangleq 0 + h(s'))$ // ***O: cola de prioridad*** $f \triangleq g + h$

  $C = \emptyset$ // ***Closed:*** nodos explorados

  mientras no $ColaVacia(O)$: // ***1ro el mejor:*** $s = \arg\min_{n \in O} f_n$

    $s = Desencola(O)$ // desempates a favor de objetivos

    si $Objetivo(s)$ **retorna** s // solución encontrada!

    $C = C \cup \{s\}$ // s explorado

    para toda $(s, n) \in Adyacentes(G, s)$: // generación: n hijo de s

      $x = (g_s + w(s, n)) + h(n)$ // posible f_n nuevo

      si $n \notin C \cup O$: $Encola(O, n, f_n \triangleq x)$

      si no si $n \in O$ **y** $x < f_n$: $Modcola(O, n, f_n \triangleq x)$

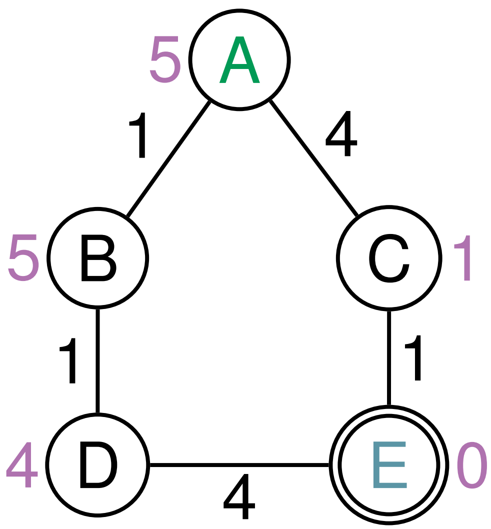
      si no si $n \in C$ **y** $x < f_n$: $C = C \setminus \{n\}$; $Encola(O, n, f_n \triangleq x)$

  retorna NULL // ninguna solución encontrada

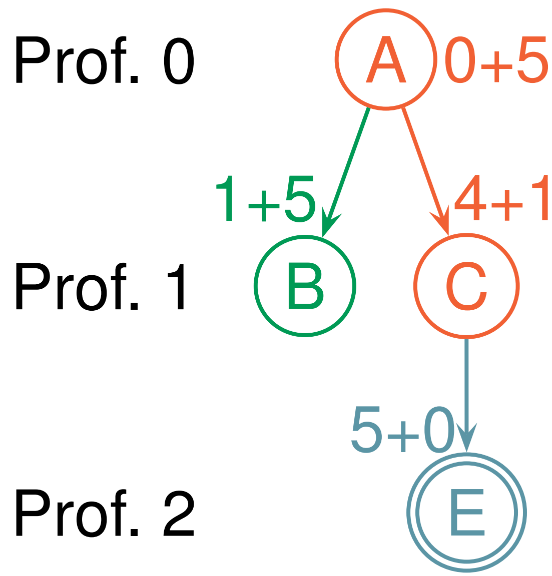
3. El árbol de búsqueda A*

Depende de $h(n)$, estimación del coste mínimo (de ir) de n a meta, $h^*(n)$:

Grafo de estados

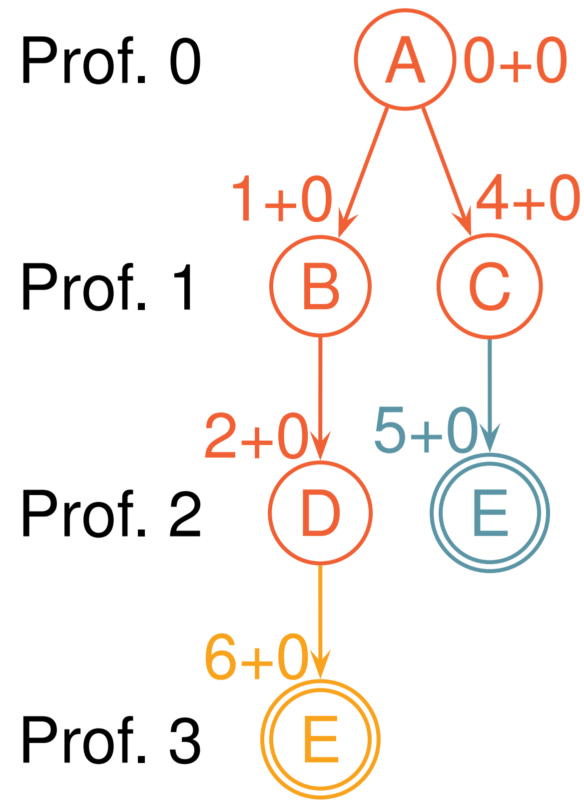


h ***a la izquierda*** ***(***h^****)***



It 3: A5C5E5B6

$h = 0$ ***(Dijkstra)***



It 5: A0B1D2C4E5

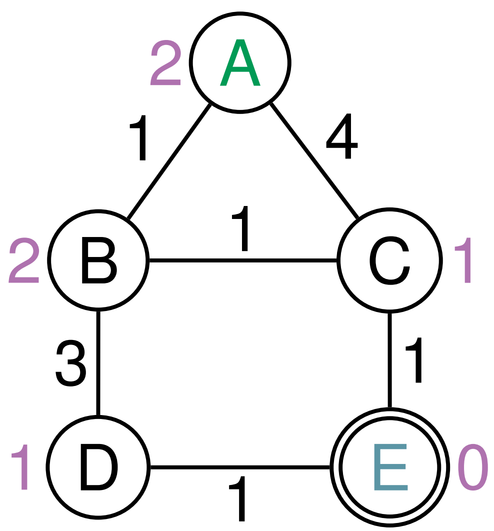
4. Optimalidad y complejidad [1, 2, 3, 4, 5, 6]

- ***Completitud:***
  - *Grafos finitos:* Sí, A* siempre acaba y es completa.
  - *Grafos infinitos:* Sí, si coste de todo camino infinito es ilimitado.
- ***Optimalidad:*** Si h es admisible, A* devuelve una solución óptima; también se dice que ***A* es admisible***.
- ***Si h es consistente***, los nodos se seleccionan para una expansión en orden no decreciente de f, mediante caminos óptimos ($g = g^*$).
  - A* no re-abre nodos cerrados (no hay que implementarlo).
  - A* equivale a Dijkstra con ***costes reducidos*** [6].
  - A* es ***óptimamente eficiente*** para h, es decir, ningún otro algoritmo con la misma h expandirá menos nodos [6].
- ***Complejidad:*** Como la de Dijkstra, si h es consistente [6].

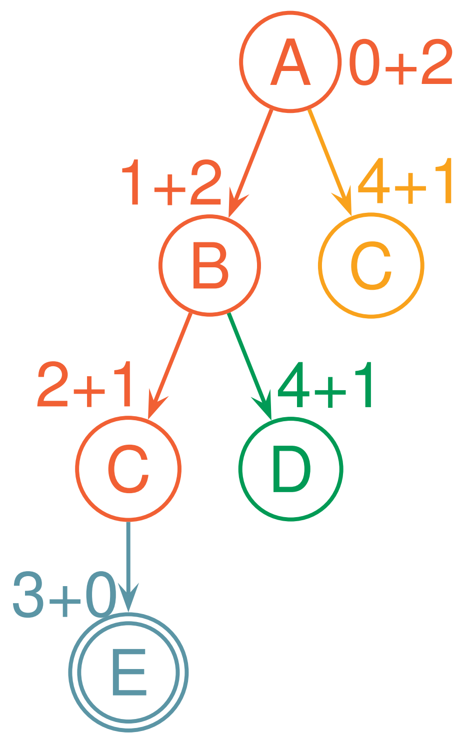
5. A* con búsqueda en árbol

A* con búsq. en árbol [5] consiste en "olvidarse" de nodos cerrados (manteniendo $C = \emptyset$) i reabrirlos como si fueran inexplorados:

Grafo de estados

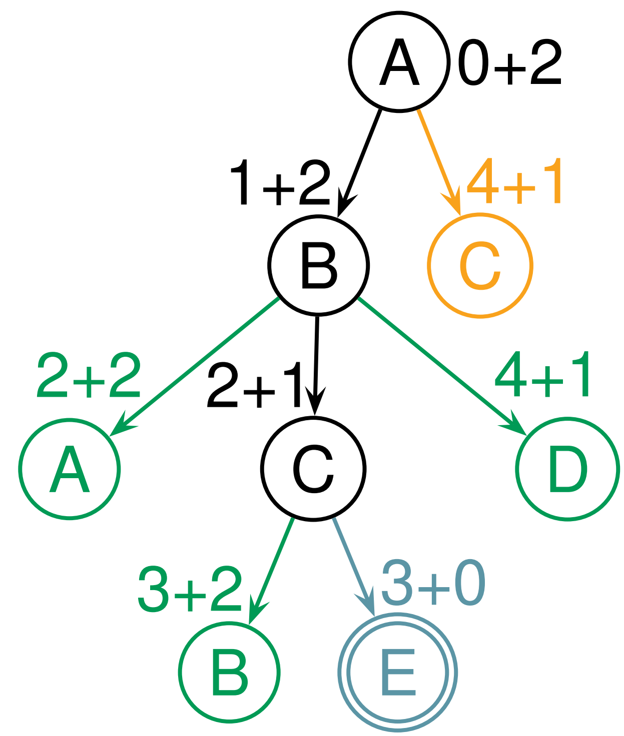


Búsq. en grafo



4: A2B3C3E3D5

Búsqueda en árbol



4: A2B3C3E3A4B5D5

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

6. Conclusiones

Hemos visto:

- El algoritmo de búsqueda A*.
- El árbol de búsqueda A*.
- La optimalidad y complejidad de búsqueda A*.
- A* con búsqueda en árbol (sin nodos cerrados).

Algunos aspectos a destacar sobre A*:

- Completa y óptima con aristas de coste positivo y h admisible.
- Más simple y eficiente si h consistente (no reexpande cerrados).
- Coste espacial excesivo, sobre todo con soluciones profundas.
- Coste espacial reducible con búsqueda en árbol.

Referencias

[1] P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 1968.

[2] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[3] R. Dechter and J. Pearl. Generalized Best-First Search Strategies and the Optimality of A*. *Journal of the ACM*, 1985.

[4] R. C. Holte. Common Misconceptions Concerning Heuristic Search. In *Proc. of SOCS-10*, 2010.

[5] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, third edition, 2010.

[6] S. Edelkamp and S. Schrödl. *Heuristic Search – Theory and Applications*. Academic Press, 2012.

astar.py

```python
#!/usr/bin/env python3
import heapq
G={'A':[('B',1),('C',4)],'B':[('A',1),('D',1)],
	'C':[('A',4),('E',1)],'D':[('B',1),('E',4)],
	'E':[('C',1),('D',4)]}
h0={'A':0,'B':0,'C':0,'D':0,'E':0}
hstar={'A':5,'B':5,'C':1,'D':4,'E':0}
def astar(G,s,t,h):
	Od={s:0}; Cd={} # Open and Closed g dict
	Oh=[]; heapq.heappush(Oh,(h[s],s,[s])) # Open heap
	while Od:
		s=None
		while s not in Od: fs,s,path=heapq.heappop(Oh) # delete-min
		gs=Od[s]
		if s==t: return gs,path
		del Od[s]; Cd[s]=gs
		for n,wsn in G[s]:
			gn=gs+wsn
			if n in Cd:
				if gn<Cd[n]: del Cd[n] # h inconsistent!
				else: continue
			elif n in Od and gn>=Od[n]: continue
			Od[n]=gn; heapq.heappush(Oh,(gn+h[n],n,path+[n]))
print(astar(G,'A','E',h0))
print(astar(G,'A','E',hstar))
```

astar.py.out

```
(5, ['A', 'C', 'E'])
(5, ['A', 'C', 'E'])
```

astar_pqdict.py

```python
#!/usr/bin/env python3
from pqdict import pqdict
G={'A':[('B',1),('C',4)],'B':[('A',1),('D',1)],
   'C':[('A',4),('E',1)],'D':[('B',1),('E',4)],
   'E':[('C',1),('D',4)]}
h0={'A':0,'B':0,'C':0,'D':0,'E':0}
hstar={'A':5,'B':5,'C':1,'D':4,'E':0}
def astar(G,s,t,h):
  O=pqdict({s:(0,h[s],[s])},key=lambda x:x[0]+x[1]); C={}
  while O:
    s,(gs,hs,path)=O.popitem()
    if s==t: return gs,path
    C[s]=gs,hs
    for n,wsn in G[s]:
      gn=gs+wsn
      if n in C:
        if gn>=C[n][0]: continue
        ogn,ohn=C[n]; del C[n]; O[n]=gn,ohn,path+[n]
      elif n in O:
        if gn>=O[n][0]: continue
        ogn,ohn,opath=O[n]; O[n]=gn,ohn,path+[n]
      else: O[n]=gn,h[n],path+[n]
print(astar(G,'A','E',h0))
print(astar(G,'A','E',hstar))
```

astar_pqdict.py.out

```
(5, ['A', 'C', 'E'])
(5, ['A', 'C', 'E'])
```