UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cerca A* [1]

Albert Sanchis
Alfons Juan

DSIC
Departament de Sistemes Informàtics i Computació

[1] Per a una correcta visualització, es requereix l'Acrobat Reader v. 7.0 o superior

Objectius formatius

- Aplicar l'algorisme A^*.
- Construir l'arbre de cerca A^*.
- Analitzar l'optimalitat i complexitat de cerca A^*.
- Distingir A^* estàndard de la versió sense nodes tancats.

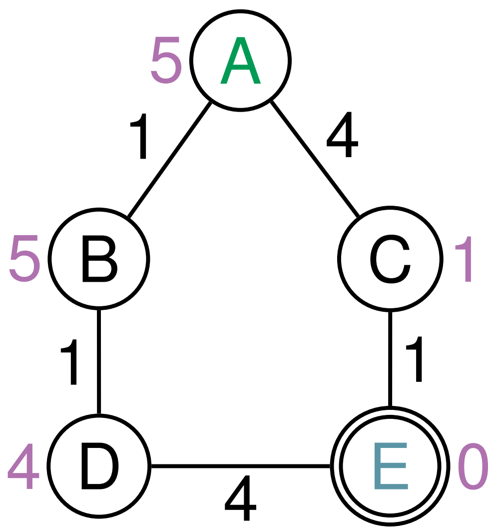
Índex

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1 Introducció

Cerca A*[*] consisteix a enumerar camins fins a trobar una solució, prioritzant els de menor cost total estimat ($f = g + h$) i evitant cicles:

Graf d'estats



Cerca

A 0+5

AB 1+5
AC 4+1

ACE 5+0 sol.!

A* generalitza UCS amb la introducció de l'***heurística*** h d'estimació del cost mínim d'arribar a meta; no negativa, nul·la en meta.

2 L'algorisme A* [1]

A* (G, s', h) // G graf ponderat, s' *start*, h ***heurística***

$O = IniCua(s', f_{s'} \triangleq 0 + h(s'))$ // ***O: cua de prioritat*** $f \triangleq g + h$

$C = \emptyset$ // ***Closed:*** nodes explorats

mentre no $CuaBuida(O)$: // ***1r el millor:*** $s = \arg\min_{n \in O} f_n$

  $s = Desencua(O)$ // desempats a favor d'objectius

  si $Objectiu(s)$ **retorna** s // solució trobada!

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

  per a tota $(s, n) \in Adjacents(G, s)$: // generació: n fill d's

    $x = (g_s + w(s, n)) + h(n)$ // possible f_n nou

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

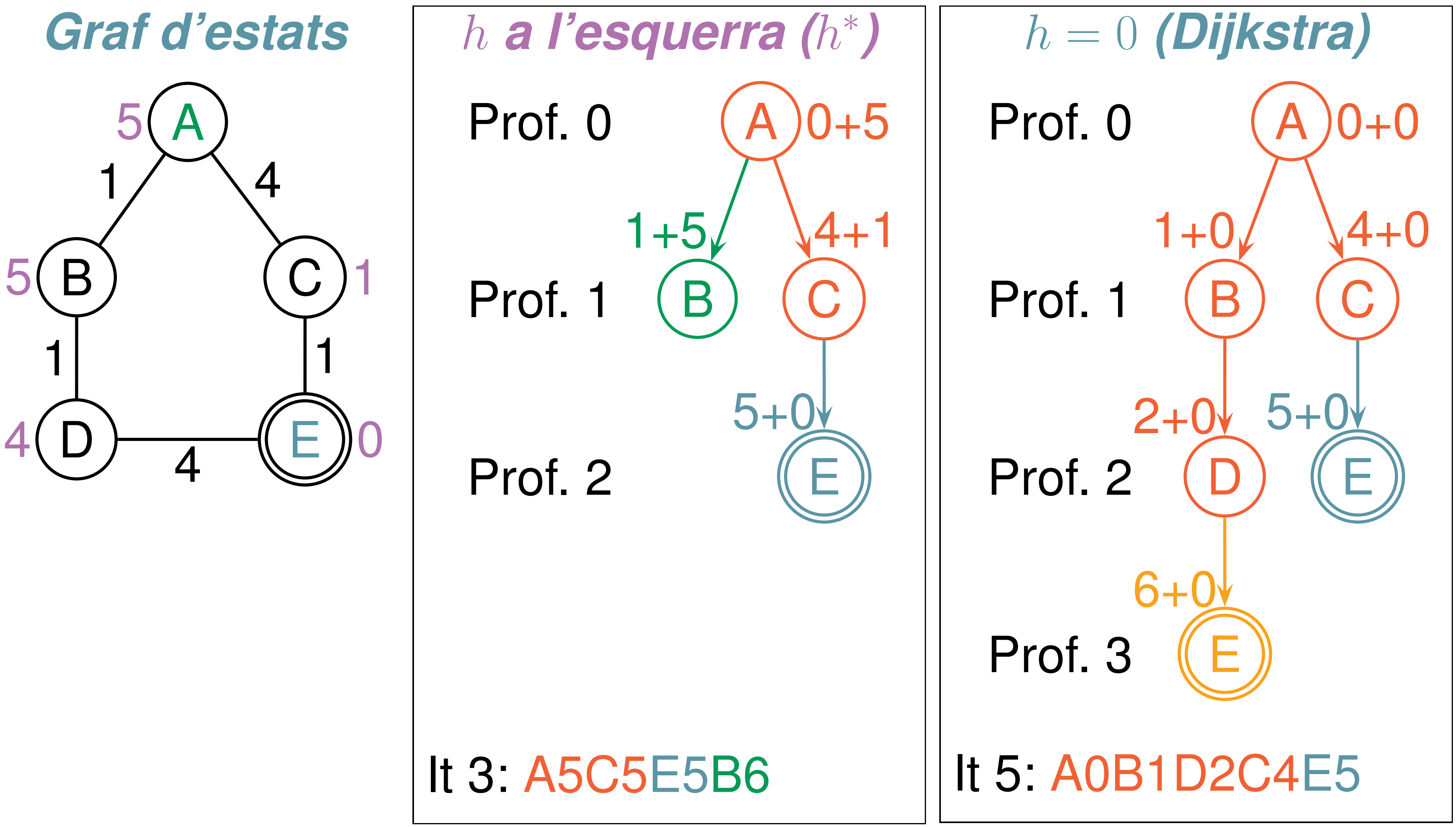
    si no si $n \in O$ **i** $x < f_n$: $Modcua(O, n, f_n \triangleq x)$

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

retorna NULL // cap solució trobada

3 L'arbre de cerca A*

Depén d'$h(n)$, estimació del cost mínim (d'anar) d'n a meta, $h^*(n)$:



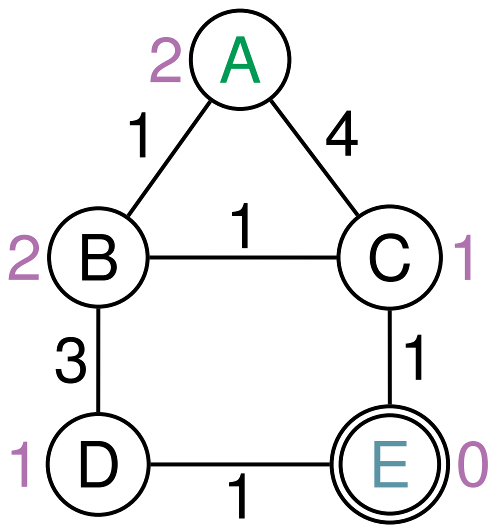
4 Optimalitat i complexitat [1, 2, 3, 4, 5, 6]

- ***Completesa:***
  - *Grafs finits:* Sí, A* sempre acaba i és complet.
  - *Grafs infinits:* Sí, si el cost de tot camí infinit és il·limitat.
- ***Optimalitat:*** Si h és admissible, A* torna una solució òptima; també es diu que ***A* és admissible***.
- ***Si h és consistent***, els nodes se seleccionen per a expansió en ordre no decreixent d'f, mitjançant camins òptims ($g = g^*$).
  - A* no re-obri nodes tancats (no cal implementar-ho).
  - A* equival a Dijkstra amb ***costs reduïts*** [6].
  - A* és ***òptimament eficient*** per a h, és a dir, cap altre algorisme amb la mateixa h expandirà menys nodes [6].
- ***Complexitat:*** Com la de Dijkstra si h és consistent [6].

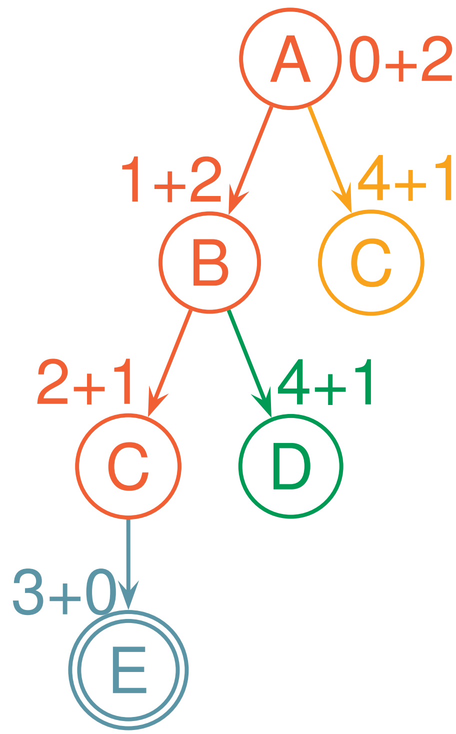
5 A* amb cerca en arbre

A* amb cerca en arbre [5] consisteix en "oblidar-se" dels nodes tancats (mantenint $C = \emptyset$) i reobrir-los com si foren inexplorats:

Graf d'estats

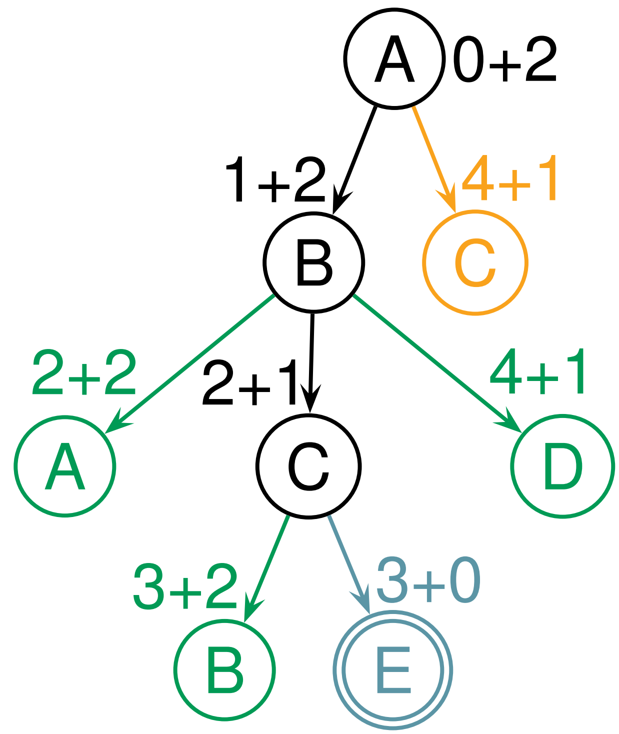


Cerca en graf



4: A2B3C3E3D5

Cerca en arbre



4: A2B3C3E3A4B5D5

6 Conclusions

Hem vist:

- L'algorisme de cerca A*.
- L'arbre de cerca A*.
- L'optimalitat i complexitat de cerca A*.
- A* amb cerca en arbre (sense nodes tancats).

Alguns aspectes a destacar sobre A*:

- Completa i òptima amb arestes de cost positiu i h admissible.
- Més simple i eficient si h consistent (no re-expandeix tancats).
- Cost espacial excessiu, sobretot amb solucions profundes.
- Cost espacial reductible amb cerca en arbre.

Referències

[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'])
```