UNIVERSITAT POLITÈCNICA DE VALÈNCIA

A* Search

Alfons Juan
Jorge Civera

DSIC

Departament de Sistemes Informàtics i Computació

Objectives

- To apply the A* algorithm.
- To build the tree of A* search.
- To analyse the optimality and complexity of A* search.
- To distinguish between A* graph and tree search.

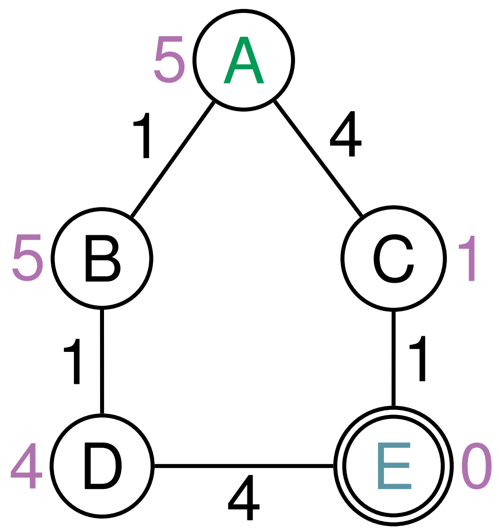
Contents

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1 Introduction

A* search consist in enumerating paths until it finds a solution, prioritising those of lowest estimated cost ($f = g + h$) and avoiding cycles:

State graph



Search

A 0+5

AB 1+5
AC 4+1

ACE 5+0 sol.!

A* generalises UCS with the incorporation of the ***heuristic*** function h that estimates the minimum cost of reaching a goal state.
h is a non-negative function, being zero at a goal state.

2 The A* algorithm (graph search) [1]

A* (G, s', h) // G weighed graph, s' *start*, h ***heuristic***

  $O = InitQueue(s', f_{s'} \triangleq 0 + h(s'))$ // ***Open: priority queue*** $f \triangleq g + h$

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

  while not $EmptyQueue(O)$: // ***best-first:*** $s = \arg\min_{n \in O} f_n$

    $s = Pop(O)$ // draws in favour of goal state

    if $Goal(s)$ **return** s // solution found!

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

    for all $(s, n) \in Adjacents(G, s)$: // generation: n is child of s

      $x = (g_s + w(s, n)) + h(n)$ // possibly new f_n

      if $n \notin C \cup O$: $Push(O, n, f_n \triangleq x)$

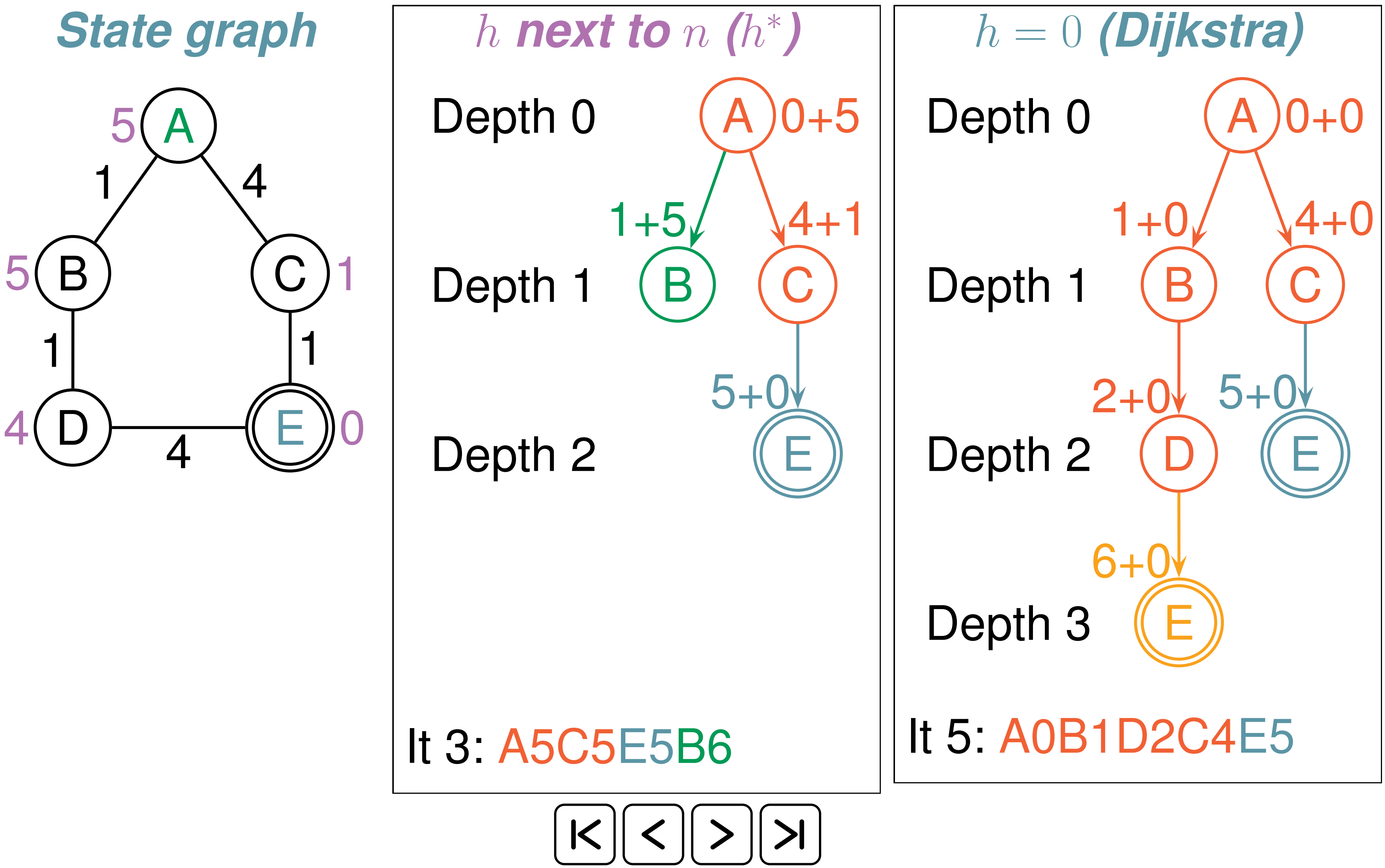
      else if $n \in O$ **and** $x < f_n$: $Update(O, n, f_n \triangleq x)$

      else if $n \in C$ **and** $x < f_n$: $C = C \setminus \{n\}$; $Push(O, n, f_n \triangleq x)$

  return NULL // solution not found

3 A* search space

It depens on $h(n)$, minimum cost estimate from n to goal state, $h^*(n)$:



4 Optimality and complexity [1, 2, 3, 4, 5, 6]

- ***Completeness:*** A* always finishes in finite graphs.
- ***Optimality:*** If h is admissible, A* returns the optimal solution; it is said that ***A* is admissible***.
- ***If h is consistent***, nodes are selected to be expanded in non-decreasing order of f, traversing optimum paths ($g = g^*$).
  - A* does not re-expand closed nodes (no need to implement it)
  - A* is equivalent to Dijkstra with ***reduced costs*** [6].
  - A* is ***optimally efficient*** for h, that is, there is no other algorithm that with the same h would expand fewer nodes [6].
- ***Complexity:*** The same as Dijkstra if h is consistent [6].

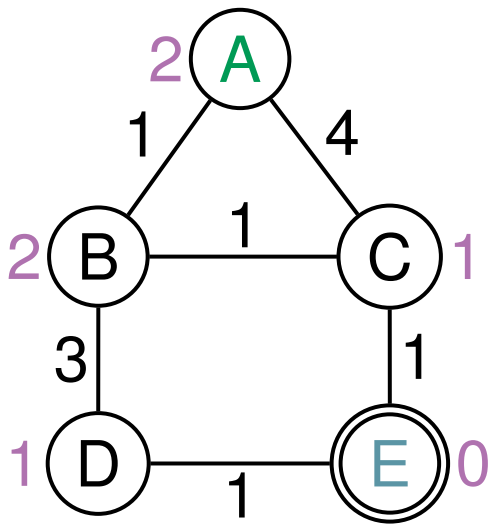
5 The A* algorithm (tree search)

A* (G, s', h)	// G weighed graph, s' *start*, h ***heuristic***
$\quad O = InitQueue(s', f_{s'} \triangleq 0 + h(s'))$	// ***Open: priority queue*** $f \triangleq g + h$
$\quad$ **while not** $EmptyQueue(O)$:	// ***best-first:*** $s = \arg\min_{n \in O} f_n$
$\qquad s = Pop(O)$	// draws in favour of goal state
$\qquad$ **if** $Goal(s)$ **return** s	// solution found!
$\qquad$ **for all** $(s, n) \in Adjacents(G, s)$:	// generation: n is child of s
$\qquad\quad x = (g_s + w(s, n)) + h(n)$	// possibly new f_n
$\qquad\quad$ **if** $n \notin O$: $Push(O, n, f_n \triangleq x)$	
$\qquad\quad$ **else if** $n \in O$ **and** $x < f_n$: $Update(O, n, f_n \triangleq x)$	
$\quad$ **return** NULL	// solution not found

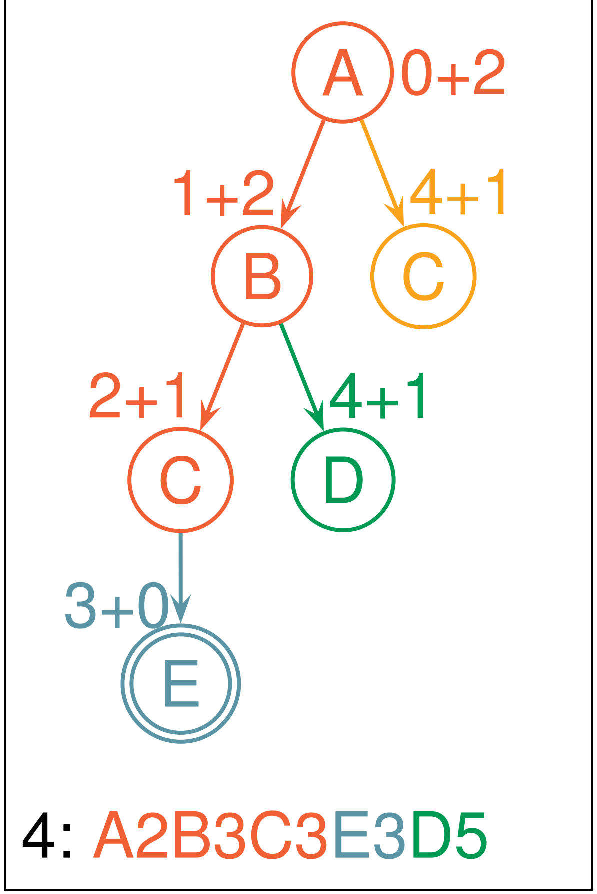
6 The A* algorithm (tree search)

A* with tree search [5] has no closed nodes ($C = \emptyset$) and re-expands them as they were new nodes:

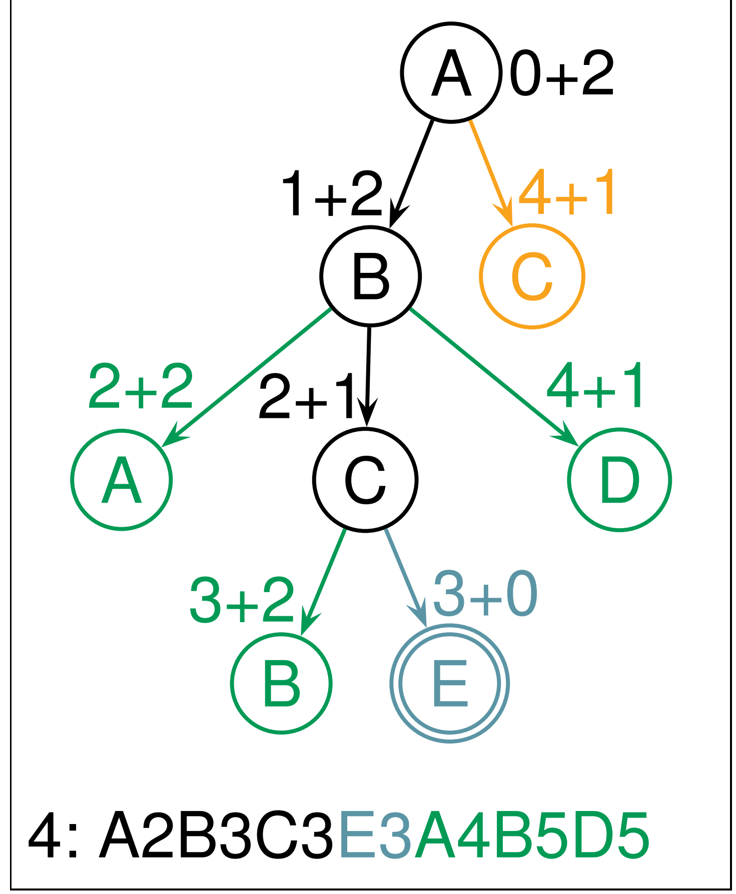
State graph



Graph search



Tree search



7 Conclusions

We have studied:

- The A* algorithm (graph search).
- The A* search space.
- Optimality and complexity in A* search.
- The A* algorithm (tree search).

Some aspects to highlight on A*:

- Complete and optimum with positive-cost edges and h admissible.
- Simpler and efficient if h consistent (closed nodes not re-expanded).
- Excessive spatial cost, especially with deep solutions.
- Reduced spatial cost with tree search.

References

[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

```
#!/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'])
```