Uniform-cost search: Dijkstra's algorithm

Alfons Juan
Jorge Civera

DSIIC

Departament de Sistemes Informàtics i Computació

Learning objectives

- To describe uniform-cost search or Dijkstra's algorithm.
- To build uniform-cost search tree.
- To analyze the optimality and complexity of uniform-cost search.

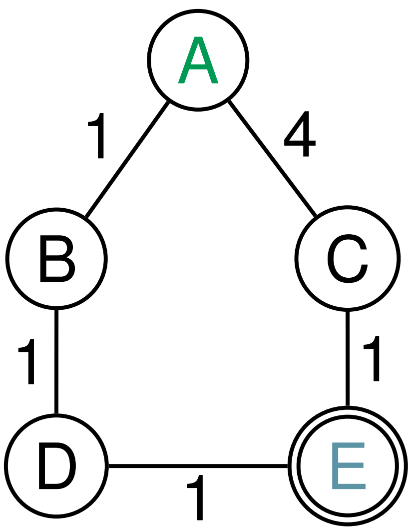
Contents

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1 Introduction

Uniform-cost search (UCS) or ***Dijkstra's algorithm*** enumerates paths until finding a solution by prioritizing paths with minimum (partial) cost and avoiding cycles:

State graph



Search

A 0

AB 1

AC 4

ABD 2

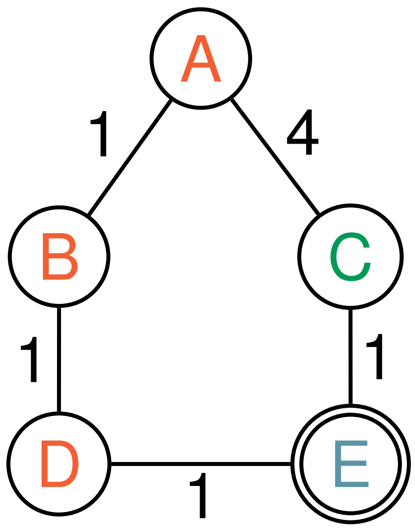
ABDE 3 sol.!

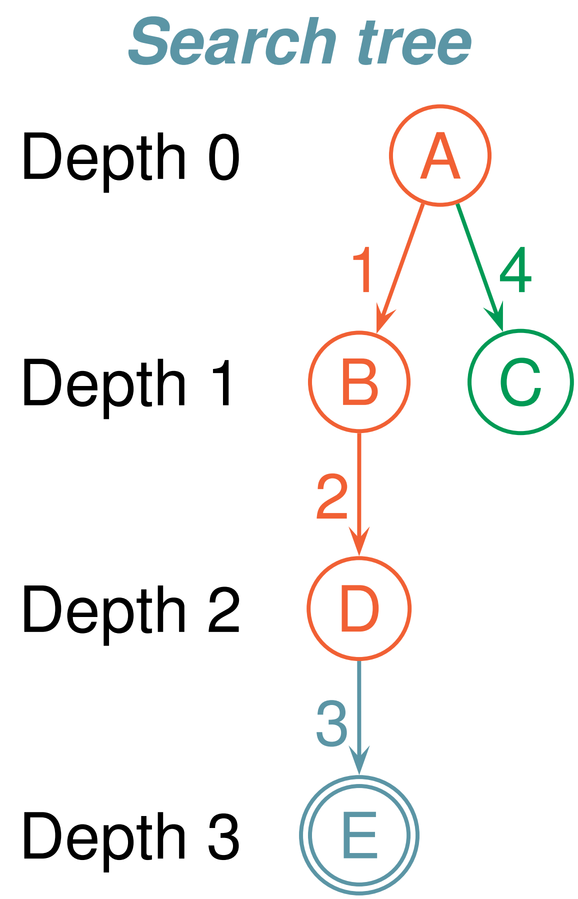
Note: UCS generalises BFS to edges of different costs.

2 Uniform-cost or Dijkstra's algorithm [1, 2, 3]

```
UCS(G, s')            //Uniform-cost search; G weighted graph, s' start
  O = InitQueue(s', g_s' ≜ 0)                // Open: priority queue g
  C = ∅                                      // Closed: explored nodes
  while not EmptyQueue(O):                   // best-first: s = argmin_{n∈O} g_n
    s = Pop(O)                               // ties solved in favor of goals
    if Goal(s) return s                      // solution found!
    C = C ∪ {s}                              // s explored
    forall (s,n) ∈ Adjacents(G,s):           // generation: n child of s
      x = g_s + w(s,n)                       // path cost from s' to n through s
      if            n ∉ C ∪ O: Push(O, n, g_n ≜ x)
      else if n ∈ O and x < g_n: Update(O, n, g_n ≜ x)
  return NULL                                // no solution found
```

3 Uniform-cost search tree





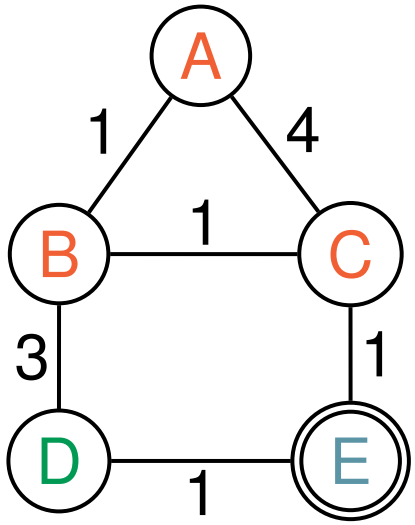
It. 4: A0B1D2E3C4 E3 sol.!

|< < > >|

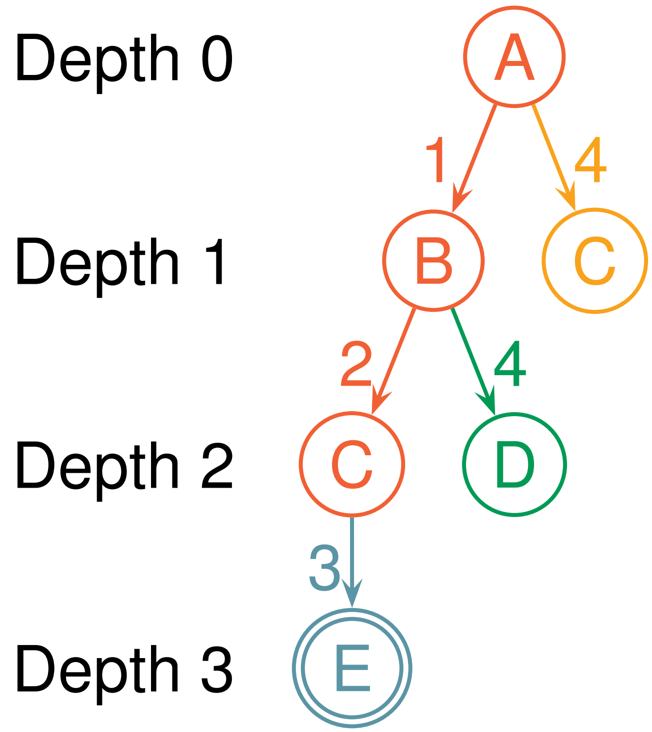
Nota: BFS returns ACE (cost 5) instead of ABDE (cost 3).

Uniform-cost search tree (cont.)

State graph



Search tree



It. 4: A0B1C2E3D4 E3 sol.!

Nota: UCS keeps track of the ***shortest paths*** from the source node to each open node, ***traversing explored nodes only.***

4 Optimality and complexity

- ***Optimality:*** Yes, with non-negative weights.

- ***Complexity:***
  - ▷ $G = (V, E)$ *explicit:* $O(|E| \log|V|)$ with a *heap* [4].
  - ▷ G *implicit* with ***branching factor*** b***:***

    Worst case: solution at depth $d = \left\lfloor \frac{C^*}{\epsilon} \right\rfloor$, where ϵ is the minimum weight and C^* is the optimal path cost.

    A full search tree is generated with nodes at depth $d + 1$.

    $O(b^{d+1})$ time and space.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

5 Conclusions

Topics covered:

- Uniform-cost search algorithm or Dijkstra's algorithm.
- Uniform-cost search tree.
- Uniform-cost search quality and complexity.

Highlights:

- Complete and optimal with positive edge costs.
- Excessive space complexity, specially with deep solutions.
- Dijkstra's algorithm is the main technique to search for shortest paths in an explicit graph; either the shortest path between two nodes, or all shortest paths between a node and the rest.

References

[1] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1959.

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

[3] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2018.

[4] Mo Chen et al. Priority Queues and Dijkstra's Algorithm. Technical report, UTCS TR-07-54, 2007.

ucs.py

```python
#!/usr/bin/env python3
import heapq
G1={'A':[('B',1),('C',4)],'B':[('A',1),('D',1)],
		'C':[('A',4),('E',1)],'D':[('B',1),('E',1)],
		'E':[('C',1),('D',1)]}
G2={'A':[('B',1),('C',4)],'B':[('A',1),('C',1),('D',3)],
		'C':[('A',4),('B',1),('E',1)],'D':[('B',3),('E',1)],
		'E':[('C',1),('D',1)]}
def ucs(G,s,t):
	Od={s:0}; Cd={} # Open and Closed g dict
	Oh=[]; heapq.heappush(Oh,(0,s,[s])) # Open heap
	while Od:
		s=None
		while s not in Od: gs,s,path=heapq.heappop(Oh) # delete-min
		if s==t: return gs,path
		del Od[s]; Cd[s]=gs
		for n,wsn in G[s]:
			gn=gs+wsn
			if n not in Cd and (n not in Od or gn<Od[n]):
				heapq.heappush(Oh,(gn,n,path+[n])) # insert
				Od[n]=gn
print(ucs(G1,'A','E'))
print(ucs(G2,'A','E'))
```

ucs.py.out

```
(3, ['A', 'B', 'D', 'E'])
(3, ['A', 'B', 'C', 'E'])
```