

Problem Solving with AI Techniques

Uninformed Search

Paul Weng

UM-SJTU Joint Institute

VE593, Fall 2018

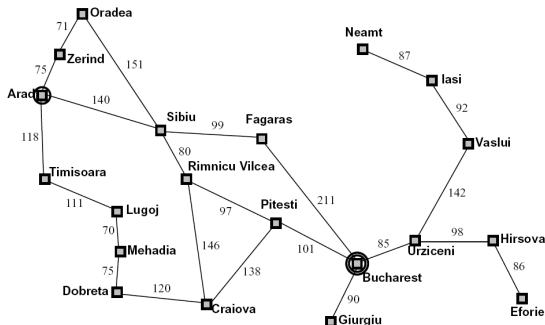


JOINT INSTITUTE
交大密西根学院

- 1 Example
- 2 Breadth-First Search
- 3 Uniform-Cost Search
- 4 Depth-First Search
- 5 Depth-Limited Search
- 6 Iterative Deepening Search
- 7 Birectional Search
- 8 Summary of Algorithms

Traveling in Romania

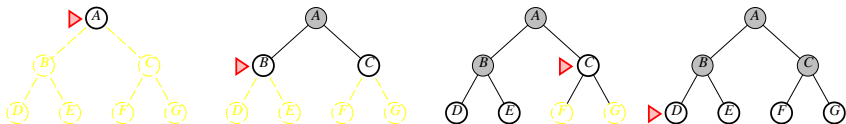
- State space
 - Cities
- Successor function
 - Roads: Go to adjacent city with cost = distance
- Start state
 - Arad
- Goal test:
 - is state = Bucharest?
- Solution?



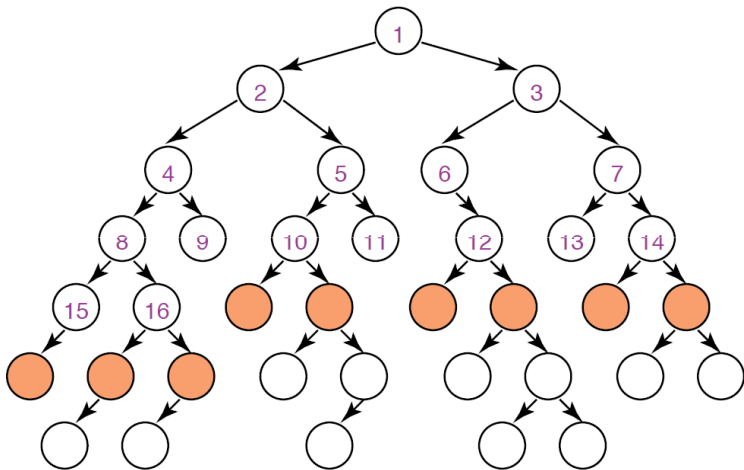
- 1 Example
- 2 **Breadth-First Search**
- 3 Uniform-Cost Search
- 4 Depth-First Search
- 5 Depth-Limited Search
- 6 Iterative Deepening Search
- 7 Birectional Search
- 8 Summary of Algorithms
- 9 Caveat

Principle of Breadth-First Search (BFS)

- **Strategy:** expand shallowest unexpanded node
- **Implementation:** fringe = queue (FIFO); put successors at end



Other Illustrative Example of Breadth-First Search



Properties of Breadth-First Search

- **Complete?** Yes (if branching factor b is finite)
- **Time?** $\mathcal{O}(1 + b + b^2 + \dots + b^d + b(b^d - 1)) = \mathcal{O}(b^{d+1})$
exponential in d
- **Space?** $\mathcal{O}(b^{d+1})$, keeps every node in memory
- **Optimal?** Yes, with all equal action costs (e.g., $= 1$)

- *Space* is the bigger problem (more than time)

- 1 Example
- 2 Breadth-First Search
- 3 **Uniform-Cost Search**
- 4 Depth-First Search
- 5 Depth-Limited Search
- 6 Iterative Deepening Search
- 7 Birectional Search
- 8 Summary of Algorithms
- 9 Caveat

Principle of Uniform-Cost Search (UCS)

- Breadth-first search optimal only if path cost is non-decreasing function of depth
for all nodes n_{d-1} at depth $d - 1$ and n_d at depth d
 $cost(n_0, \dots, n_d) \geq cost(n_0, \dots, n_{d-1})$
- Can we guarantee optimality for any positive action cost?

Principle of Uniform-Cost Search (UCS)

- Breadth-first search optimal only if path cost is non-decreasing function of depth
for all nodes n_{d-1} at depth $d - 1$ and n_d at depth d
 $cost(n_0, \dots, n_d) \geq cost(n_0, \dots, n_{d-1})$
- Can we guarantee optimality for any positive action cost?
- **Strategy:** expand least-cost unexpanded node
- **Implementation:** fringe = priority queue ordered by path cost, lowest first
- Equivalent to breadth-first search if action costs all equal
- Cost-aware breadth-first search

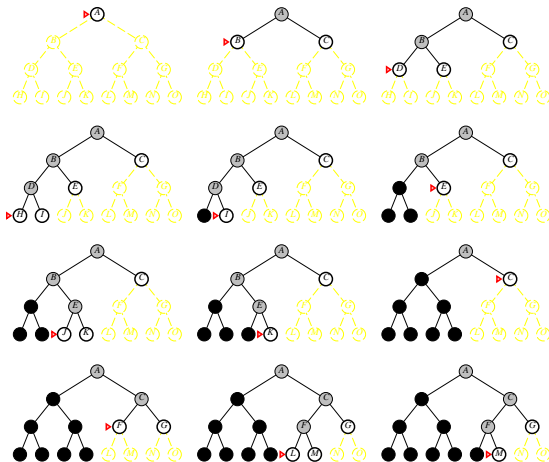
Properties of Uniform-Cost Search

- **Complete?** Yes, if action cost $\geq \varepsilon$ (i.e., strictly positive)
- **Time?** # of nodes with $g \leq$ cost of optimal solution c^* , $\mathcal{O}(b^{\lceil \frac{c^*}{\varepsilon} \rceil})$
- **Space?** # of nodes with $g \leq$ cost of optimal solution c^* , $\mathcal{O}(b^{\lceil \frac{c^*}{\varepsilon} \rceil})$
- **Optimal?** Yes, nodes expanded in increasing order of $g(n)$
where $g(n)$ cost of subpath from root to n

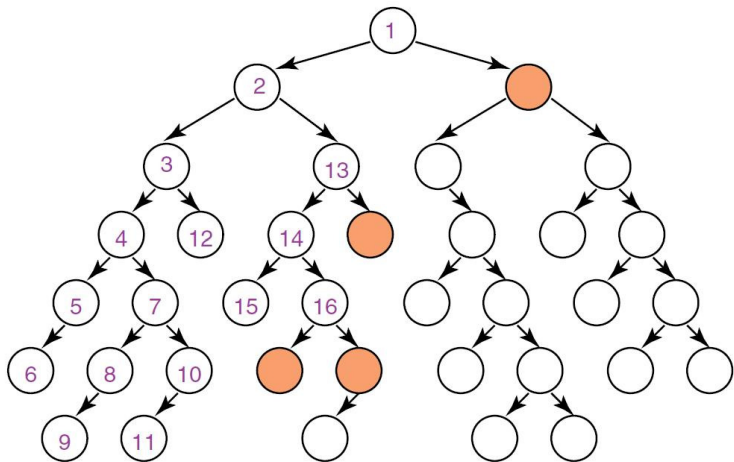
- 1 Example
- 2 Breadth-First Search
- 3 Uniform-Cost Search
- 4 Depth-First Search**
- 5 Depth-Limited Search
- 6 Iterative Deepening Search
- 7 Birectional Search
- 8 Summary of Algorithms
- 9 Caveat

Principle of Depth-First Search (DFS)

- **Strategy:** expand deepest unexpanded node
- **Implementation:** fringe = stack (LIFO); put successors at front



Other Illustrative Example of Depth-First Search



Properties of Depth-First Search

- **Complete?** No: fails in infinite-depth state-spaces and state-spaces with loops
Can be modified to avoid repeated states along path \Rightarrow complete in finite state-space graphs
- **Time?** $\mathcal{O}(b^m)$ where b is maximum branching factor of search tree
bad if $m \gg d$, but may be much faster than breadth-first search if solutions are dense
- **Space?** $\mathcal{O}(bm)$, i.e., linear space!
- **Optimal?** No

- 1 Example
- 2 Breadth-First Search
- 3 Uniform-Cost Search
- 4 Depth-First Search
- 5 Depth-Limited Search**
- 6 Iterative Deepening Search
- 7 Birectional Search
- 8 Summary of Algorithms
- 9 Caveat

Principle of Depth-Limited Search (DLS)

- Depth-first search complete only for finite state-space graphs
- Can we modify it to make it complete?
- **Idea:** depth-limited search with depth limit l , i.e., nodes at depth l are treated as having no successors
- Depth-limited search is complete if we know l such that a solution exists at depth lower or equal than l

Algorithm

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)
```

```
function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
  cutoff-occurred?  $\leftarrow$  false
  if GOAL-TEST(problem, STATE[node]) then return node
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
    result  $\leftarrow$  RECURSIVE-DLS(successor, problem, limit)
    if result = cutoff then cutoff-occurred?  $\leftarrow$  true
    else if result  $\neq$  failure then return result
  if cutoff-occurred? then return cutoff else return failure
```

- 1 Example
- 2 Breadth-First Search
- 3 Uniform-Cost Search
- 4 Depth-First Search
- 5 Depth-Limited Search
- 6 Iterative Deepening Search**
- 7 Birectional Search
- 8 Summary of Algorithms
- 9 Caveat

Principle of Iterative Deepening Search (IDS)

- What if we don't know such l ?
- **Idea:** Try them all: 0, 1, 2, 3, ...

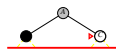
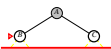
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution
  inputs: problem, a problem

  for depth  $\leftarrow$  0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result
  end
```

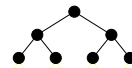
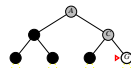
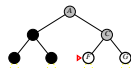
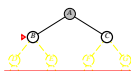
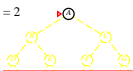
Limit = 0



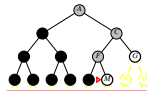
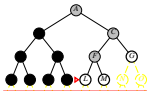
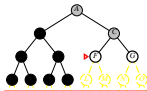
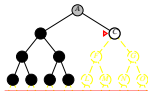
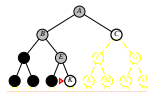
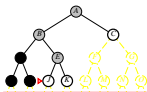
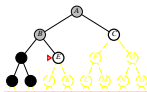
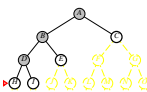
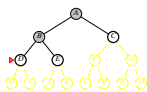
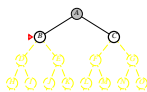
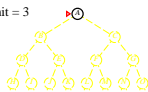
Limit = 1



Limit = 2



Limit = 3



Properties of Iterative Deepening Search

- **Complete?** Yes, if a solution exists at a finite depth
- **Time?** $\mathcal{O}((d+1)b^0 + db^1 + (d-1)d^2 + \dots + b^d) = \mathcal{O}(b^d)$
- **Space?** $\mathcal{O}(bd)$
- **Optimal?** Yes, if all equal action costs

Comparison IDS/DLS

- Number of nodes generated in a depth-limited search to depth d with branching factor b :

$$N_{DLS} = b^0 + b^1 + b^2 + \dots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth d with branching factor b :

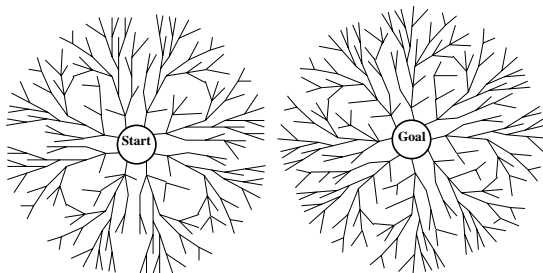
$$N_{IDS} = (d+1)b^0 + db^1 + (d-1)b^2 + \dots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

- Example: for $b = 10$, $d = 5$:
 - $N_{DLS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111$
 - $N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456$
- Overhead $= (123,456 - 111,111)/111,111 = 11\%$

- 1 Example
- 2 Breadth-First Search
- 3 Uniform-Cost Search
- 4 Depth-First Search
- 5 Depth-Limited Search
- 6 Iterative Deepening Search
- 7 Birectional Search**
- 8 Summary of Algorithms
- 9 Caveat

Principle of Bidirectional Search

- **Strategy:** run two simultaneous searches, one forward from initial state and the other backward from goal state, stop when they meet
- **Implementation:** one fringe stored as hash table



Properties of Bidirectional Search

- **Complete?** Yes, with two breadth-first searches
- **Time?** $\mathcal{O}(b^{\frac{d}{2}})$
- **Space?** $\mathcal{O}(b^{\frac{d}{2}})$
- **Optimal?** Yes, with two breadth-first searches if all action costs equal
- Space requirement still exponential
- For $d = 6$, $b = 10$, 22,200 generated nodes vs 11,111,100 for BFS
- **Issue:** How to search backward?

- 1 Example
- 2 Breadth-First Search
- 3 Uniform-Cost Search
- 4 Depth-First Search
- 5 Depth-Limited Search
- 6 Iterative Deepening Search
- 7 Birectional Search
- 8 Summary of Algorithms**
- 9 Caveat

Summary

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional
Complete	Yes ¹	Yes ^{1,2}	No	Yes ³	Yes ¹	Yes ^{1,4}
Time	b^{d+1}	$b^{\lceil \frac{c^*}{\epsilon} \rceil}$	b^m	b^l	b^d	$b^{\frac{1}{2}}$
Space	b^{d+1}	$b^{\lceil \frac{c^*}{\epsilon} \rceil}$	bm	bl	bd	$b^{\frac{1}{2}}$
Optimal?	Yes ⁵	Yes	No	No	Yes ⁵	Yes ^{4,5}

¹If b finite

²If costs > 0

³If $l \geq d$

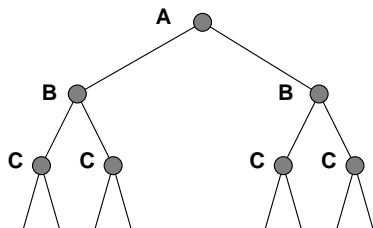
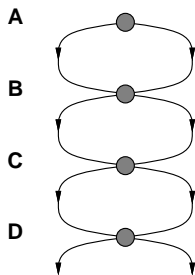
⁴If both directions use breadth-first search

⁵If costs identical

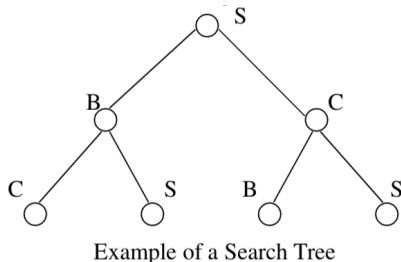
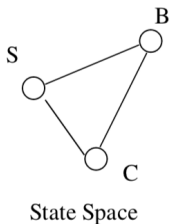
- 1 Example
- 2 Breadth-First Search
- 3 Uniform-Cost Search
- 4 Depth-First Search
- 5 Depth-Limited Search
- 6 Iterative Deepening Search
- 7 Birectional Search
- 8 Summary of Algorithms
- 9 **Caveat**

Repeated States

- Failure to detect repeated states can turn a linear problem into an exponential one!



Repeated States: Cycles



- During search, never regenerate a visited state
 - must keep track of all visited states (needs a lot of memory)
 - approximation for DFS/DLS: only avoid states in a fixed limited memory
 - optimal for BFS and UCS, not for DFS

Graph Search

function GRAPH-SEARCH(*problem*, *fringe*) **returns** a solution, or failure

closed \leftarrow an empty set

fringe \leftarrow INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)

loop do

if *fringe* is empty **then return** failure

node \leftarrow REMOVE-FRONT(*fringe*)

if GOAL-TEST(*problem*, STATE[*node*]) **then return** *node*

if STATE[*node*] is not in *closed* **then**

 add STATE[*node*] to *closed*

fringe \leftarrow INSERTALL(EXPAND(*node*, *problem*), *fringe*)

end