UNINFORMED SEARCH

Fundamentals of Artificial Intelligence

Session 08

Pramod Sharma
pramod.sharma@prasami.com

2

Agenda

- Introduction
- Breadth-first search
- Depth-first search
- Bidirectional search

4/11/2024

pra-sâmi

3

Informed search

- Guided search
- Search with knowledge
- Quick Solution
- Less Complex
- Best First Search, Greedy Best search, A*, etc
- A cleverer strategy that searches toward the goal, based on the information from the current state so far

4/11/2024

4

Uninformed Search

No prior knowledge

Time consuming

Search without information

More complex

Two forms namely;
- Breadth first
- Depth first

No information about:
- The number of steps
- The path cost from the current state to the goal

4/11/2024

5

Uninformed Search Strategies

- Breadth-first search
  - Uniform cost search
- Depth-first search
  - Depth-limited search
  - Iterative deepening search
- Bidirectional search

pra-sâmi

6

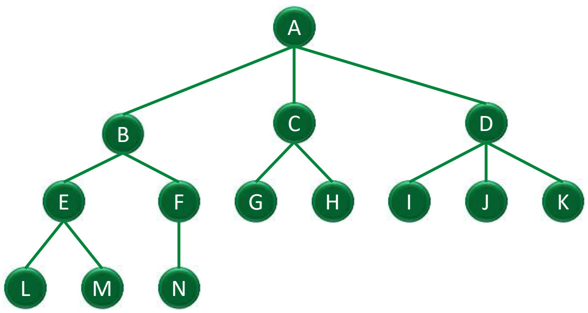
Breadth-first Search

- Breadth-first search is the most common search strategy for traversing a tree or graph
  - Searches breadthwise in a tree or graph
- It's a FIFO technique
  - Implements a queue
  - The root node is expanded first; Thereafter children of root node; Then their successors and so on…
- Level search technique
  - Moves level by level
- New nodes are getting added to the queue
- Goal test is applied to each node when it is generated rather than when it is selected for expansion
- For graph search, discards any new path to a state already in the frontier or explored set
  - Breadth-first search always has the shallowest path to every node on the frontier

pra-sâmi

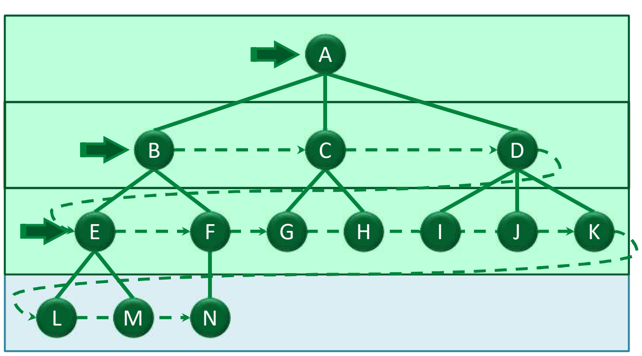
7

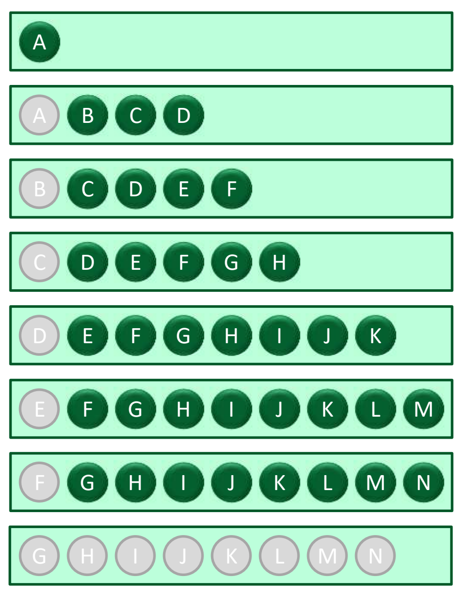
Breadth-first Search



8

Breadth-first Search



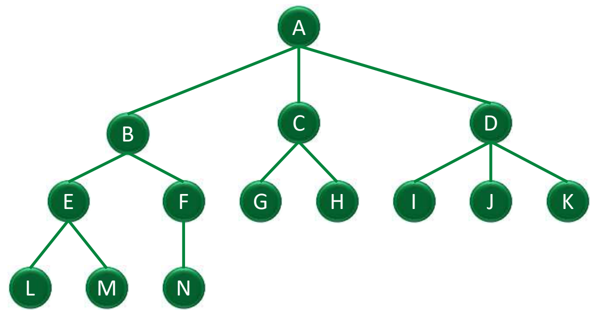


All leaf nodes…

9

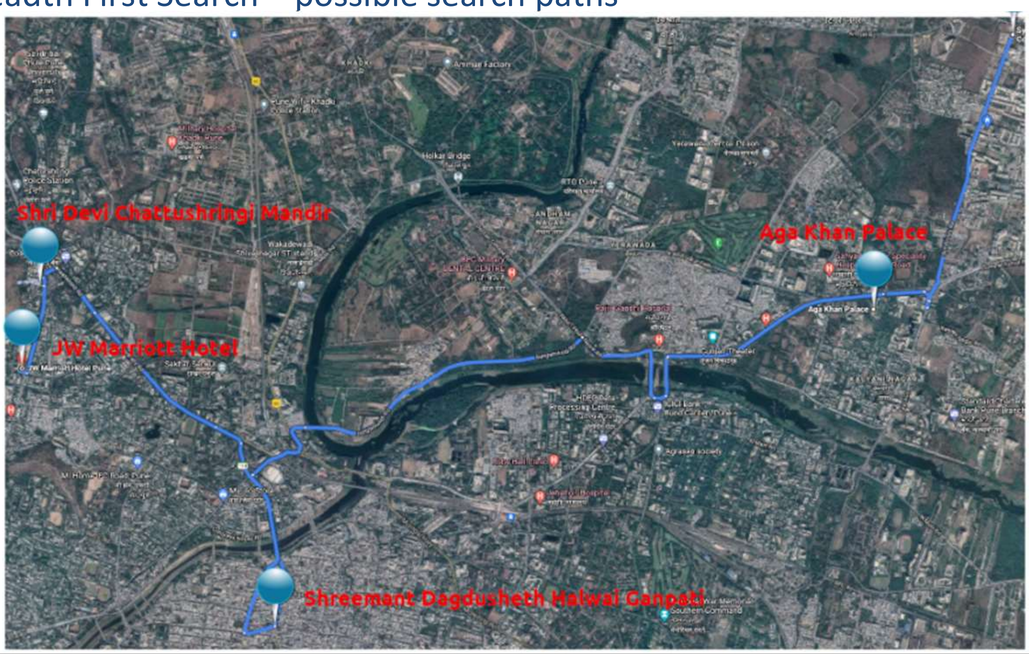
Breadth-First Search

- Complete
  - Has to search all nodes
  - Finds the solution eventually
- Optimal: yes, if step cost is 1
- Time Complexity : O(b^d)
  - b: branch factor
  - d : depth
- Space Complexity:
  - Memory size of frontier which is O (b^d)



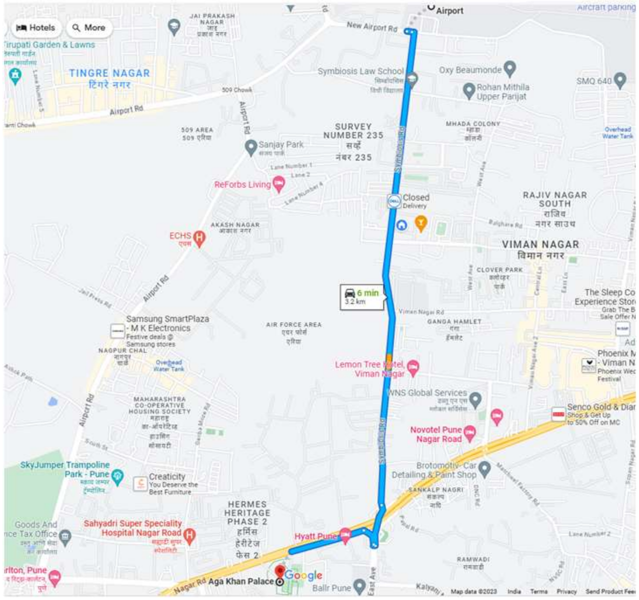
10

Breadth First Search – possible search paths



11

Breadth First Search – possible search paths



pra-sâmi

12

Performance Heavy

- An exponential complexity bound such as O (b^d) is scary
- Table with branching factor b = 10,
  - 1 million nodes can be generated per second
  - A node requires 1000 bytes of storage
- Two major issues
  - The memory requirements are a bigger problem for breadth-first search than is the execution time
  - 13 days for a problem with search depth 12 may still be ok; but One Petabyte of memory
  - Time is still a major factor
  - If a problem has a solution at depth 16, then it will take about 350 years
  - Use current uninformed methods for any small instances

Depth	Nodes	Time	Memory
2	110	0.11 mili sec	107 KB
4	11110	11 mili sec	10.6 MB
6	10^6	1.1 sec	1 GB
8	10^8	2 min	103 GB
10	10^{10}	3 hour	10 TB
12	10^{12}	13 days	1 Peta B
14	10^{14}	3.5 years	99 Peta B
16	10^{16}	350 years	10 Exa B

Estimated values on modern laptops

pra-sâmi

13

Perf

- An
  O (
- Tab
  - 1
  - A
- Two
  - Th
    br
  - 13
    st
  - Ti
  - If
    ta
  - Us
    in

mory
7 KB
6 MB
GB
3 GB
0 TB
eta B
Peta B
Exa B

ps

pra-sâmi

14

Advantage - Disadvantages

Advantages	Disadvantage
❑ BFS will provide a solution if any solution exists. ❑ If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.	❑ It requires lots of memory since each level of the tree must be saved into memory to expand the next level. ❑ BFS needs lots of time if the solution is far away from the root node.

pra-sâmi

15

Breadth-first Search – Pseudo Code

- Create two empty **queues**
- Start from the initial node and add it to the ordered open queue
- Following steps are repeated until the final node or endpoint is reached
  - If the open queue is empty exit the loop and return a False statement which says that the final node cannot be reached
  - Select the top node in the open queue and move it to the closed queue while keeping track of the parent node
  - If the node removed is the endpoint node return a True statement meaning a path has been found and moving the node to the closed queue
  - However if it is not the endpoint node then list down all the neighboring nodes of it and add them to the open queue

4/11/2024

pra-sâmi

16

Uniform Cost Search

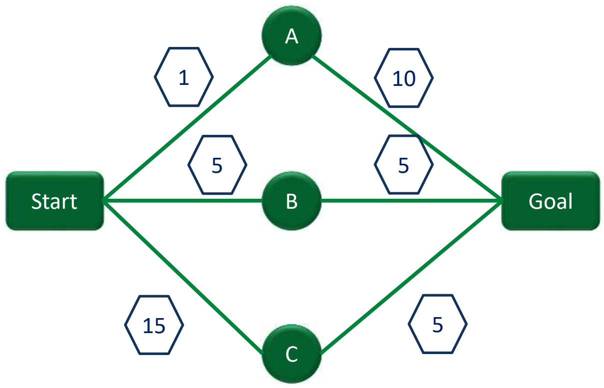
- Breadth-first finds the shallowest goal state
  - But not necessarily be the least-cost solution
  - Work only if all step costs are equal
- Uniform cost search
  - Modifies breadth-first strategy
    - By always expanding the lowest-cost node
  - The lowest-cost node is measured by the path cost g(n) i.e. cost to reach the **node**
- The goal test is applied to a node when it is selected for expansion
- A test is added in case a better path is found to a node currently on the frontier

4/11/2024

pra-sâmi

17

Uniform Cost Search

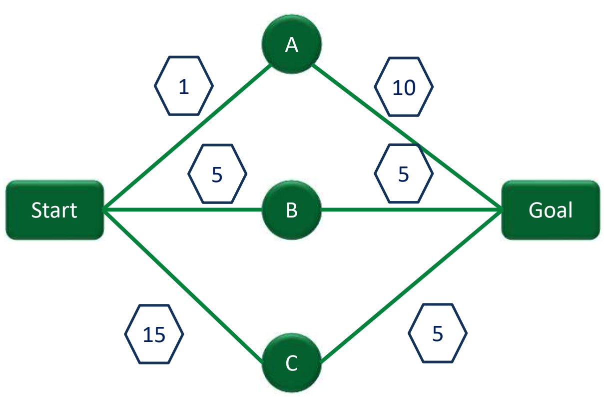


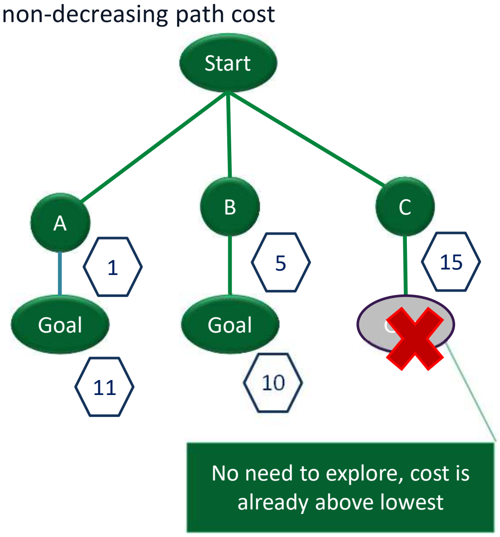
pra-sâmi

18

Uniform Cost Search

- The first found solution is guaranteed to be the cheapest in non-decreasing path cost
  - Least in depth
  - But restrict to non-decreasing path cost
  - Unsuitable for operators with negative cost (recoveries)





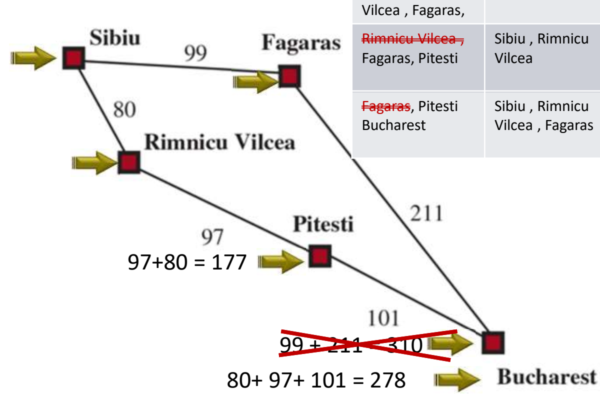
pra-sâmi

19

Back to Romania

- The successors of Sibiu are Rimnicu Vilcea and Fagaras,
- The least-cost node, Rimnicu Vilcea, is expanded next adding Pitesti with cost 177
- The least-cost node is now Fagaras
  - Adding Bucharest with cost 99 + 211 = 310
  - A goal node has been generated
- Uniform-cost search keeps going
- Choosing Pitesti adds a second path to Bucharest with cost 80+ 97+ 101 = 278.
- New path is better than the old one;

Open	Close
Sibiu	
~~Sibiu~~ , Rimnicu Vilcea , Fagaras,	Sibiu
~~Rimnicu Vilcea ,~~ Fagaras, Pitesti	Sibiu , Rimnicu Vilcea
~~Fagaras~~, Pitesti Bucharest	Sibiu , Rimnicu Vilcea , Fagaras



The algorithm tests for goals only when it expands a node, not when it generates a node.

21

Uniform Cost Search

- Expand least-cost unexpanded node
  - Uniform-cost search expands nodes in order of their optimal path cost
  - Search does not care about the number of steps a path has, but only about their cost to reach the node.
- Implementation:
  - fringe = queue ordered by path cost
- Equivalent to breadth-first if step costs all equal
- Complete? Yes
- Time? # of nodes with g ≤ cost of optimal solution, O($b^{(1+\text{ceiling}(C*/\varepsilon))}$)
  - Where C* be the cost of the optimal solution, and ε is positive constant (every action cost)
- Space? # of nodes with g ≤ cost of optimal solution, O($b^{(1+\text{ceiling}(C*/\varepsilon))}$)
- Optimal? Yes – nodes expanded in increasing order of g(n)

22

Uniform Cost Search

- When all step costs are the same,
  - Uniform-cost search is similar to breadth-first search,
- Except
  - That the breadth-first search stops as soon as it visits a goal
  - Whereas uniform-cost search examines all the nodes at the goal's depth to see if one has a lower cost
  - Uniform-cost search does strictly more work by expanding nodes at depth d unnecessarily
- Advantages:
  - Uniform cost search is optimal because at every state the path with the least cost is chosen.
- Disadvantages:
  - It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop
    - E.g, a sequence of NoOp actions

4/11/2024

pra-sâmi

23

Use of Breadth First Search

- Web Crawlers
- Social networking websites - for finding the people in the specified distance (depth)
- Torrenting/peer-to-peer network to look for neighboring computers
- GPS navigation systems can use it to find nearby locations

4/11/2024

pra-sâmi

24

Depth First Search

pra-sâmi

25

Depth-first search

- Always expands one of the nodes at the deepest level of the tree
- Only when the search hits a dead end
  - Goes back and expands nodes at shallower levels
  - Dead end → leaf nodes but not the goal
- Backtracking search
  - Only one successor is generated on expansion
  - Rather than all successors
  - Lower memory
- Implementation:
  - fringe = LIFO queue, i.e., put successors at front

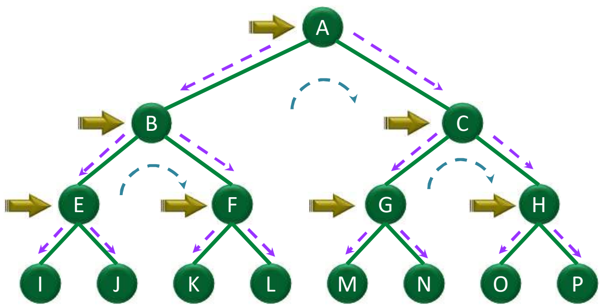
pra-sâmi

26

Depth-first search

- Expand deepest unexpanded node

- Implementation:
  - fringe = LIFO queue, i.e., put successors at front



pra-sâmi

28

Depth-first search

- Not complete
  - Because a path may be infinite or looping
  - Fails in infinite-depth spaces, spaces with loops
  - Needs modification to avoid repeated states along path
    - Complete in finite spaces

- Not optimal
  - It doesn't guarantee the best solution

- It overcomes
  - Time and space complexities

- Time? O(b^m) where m is the maximum depth of any node
  - Terrible if m is much larger than d
  - But if solutions are dense, may be much faster than breadth-first

- Space? O(b x m), i.e., linear space!

pra-sâmi

Depth-first search – Pseudo Code

- Create two empty **Stacks**
  - Open and closed
- Start from the initial node and add it to the ordered open stack
- Following steps are repeated until the final node or endpoint is reached
  - If the open stack is empty exit the loop and return a False statement which says that the final node cannot be reached
  - Select the top node in the open stack and move it to the closed stack while keeping track of the parent node
  - If the node removed is the endpoint node return a True statement meaning a path has been found and moving the node to the closed list
  - However if it is not the endpoint node then list down all the neighboring nodes of it and add them to the open stack

pra-sâmi

Depth - Limited Strategy

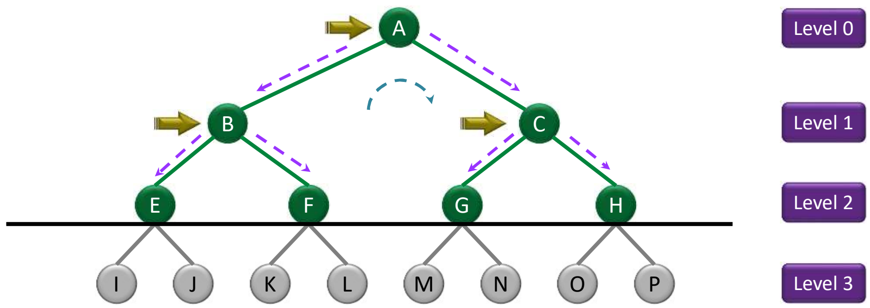
- It is depth-first search
  - With a predefined maximum depth
  - However, it is usually not easy to define the suitable maximum depth
  - Too small → no solution can be found
  - Too large → the same problems as Depth - First
- Anyway the search is
  - complete
  - but still not optimal
- Depth-first with depth cutoff k
  - maximal depth below which nodes are not expanded
  - Treat them as leaf node
- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)

pra-sâmi

31

Depth - Limited Strategy

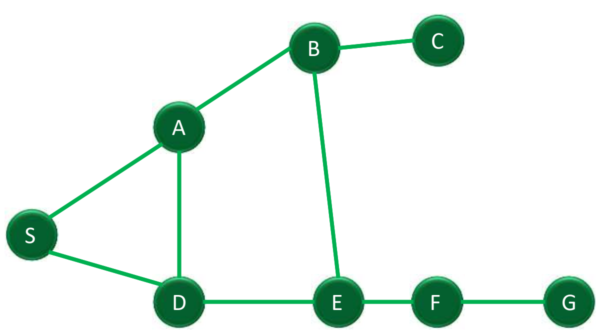
- Level 2



pra-sâmi

32

Depth - Limited Strategy

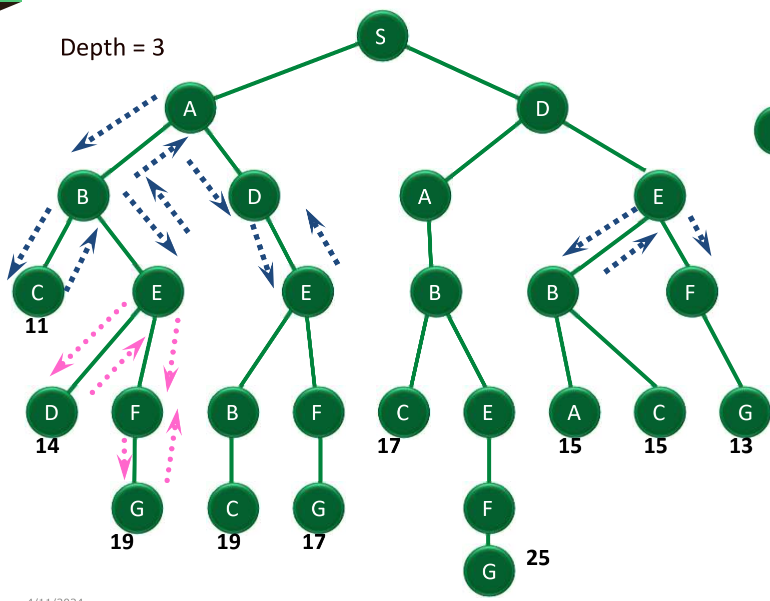


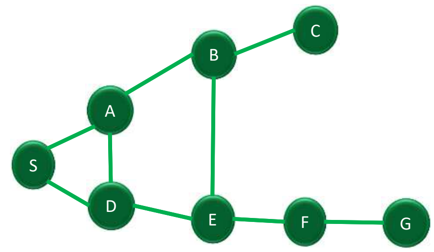
pra-sâmi

33

Depth - Limited Strategy

Depth = 3





It also introduces an additional source of incompleteness if we choose <d, that is, the shallowest goal is beyond the depth limit

pra-sâmi

34

Depth - Limited Strategy

- Advantages:
  - Depth-limited search is Memory efficient.
- Disadvantages:
  - Depth-limited search also has a disadvantage of incompleteness.
  - It may not be optimal if the problem has more than one solution.
- Completeness:
  - DLS search algorithm is complete if the solution is above the depth-limit.
- Time Complexity:
  - Time complexity of DLS algorithm is O(b^{ℓ}).
- Space Complexity: Space complexity of DLS algorithm is O(b × ℓ).
- Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d.

pra-sâmi

35

Iterative Deepening Search

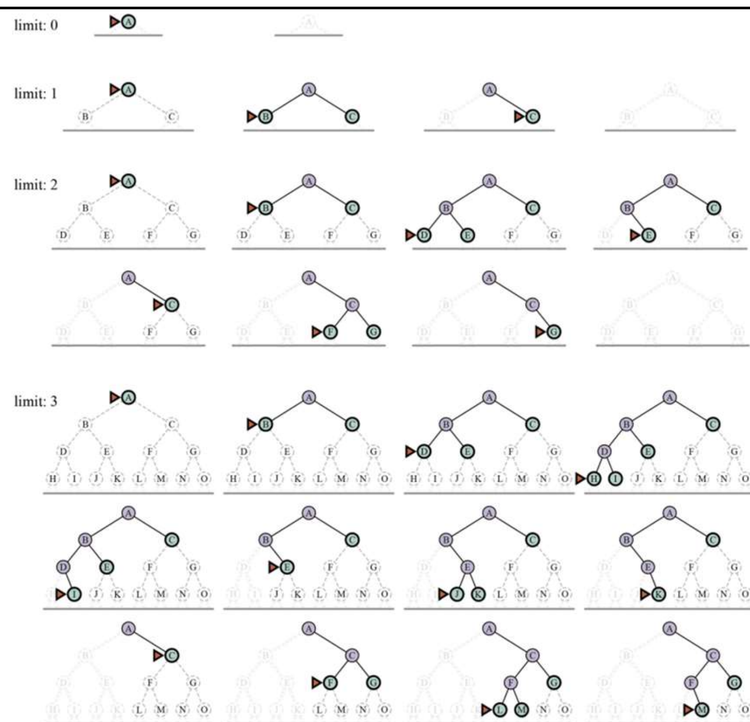
- No more choosing of the best depth limit
- It tries all possible depth limits:
  - First 0, then 1, 2, and so on
  - Combines the benefits of depth-first and breadth-first search

4/11/2024

pra-sâmi

36

Iterative Deepening Search

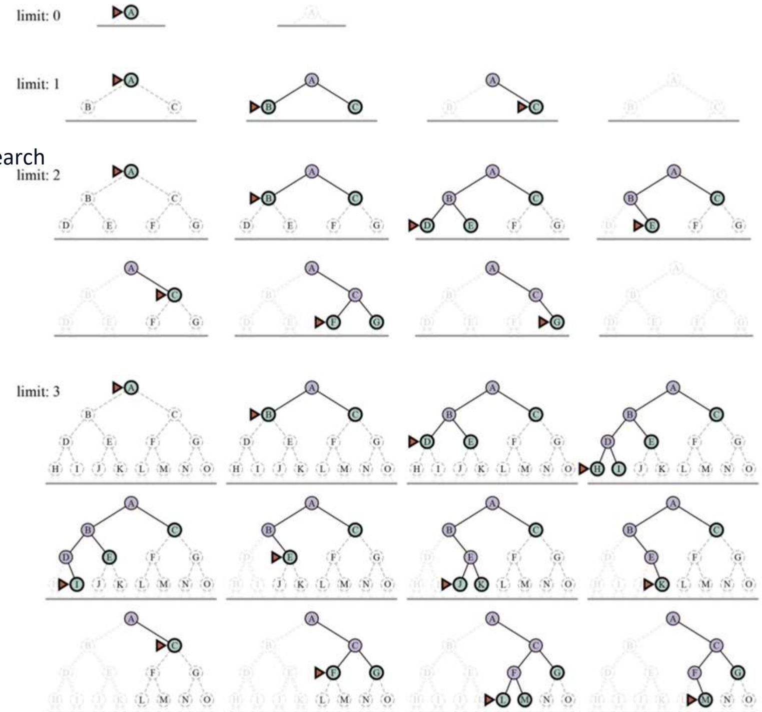


4/11/2024

37

Iterative Deepening Search

- Optimal : Yes, if step cost = 1
  - Otherwise use Iterative Lengthening search
- Complete : Yes
- Time and space complexities
  - Reasonable
- Time : $O(b^d)$
- Space: $O(b^d)$
- Suitable for the problem
  - Having a large search space
  - Depth of the solution is not known



4/11/2024

38

Iterative Lengthening Search

- Iterative Deepening Search is using depth as limit
- Iterative Lengthening Search is using path cost as limit
  - An iterative version for uniform cost search
  - Has the advantages of uniform cost search
    - while avoiding its memory requirements
  - but ILS incurs substantial overhead
    - compared to uniform cost search

4/11/2024

pra-sâmi

39

Use of Depth First Search

- Scheduling jobs from the given dependencies among jobs
- Find **a path** between two given points
- Solving puzzles fast when only one solution is needed

pra-sâmi

40

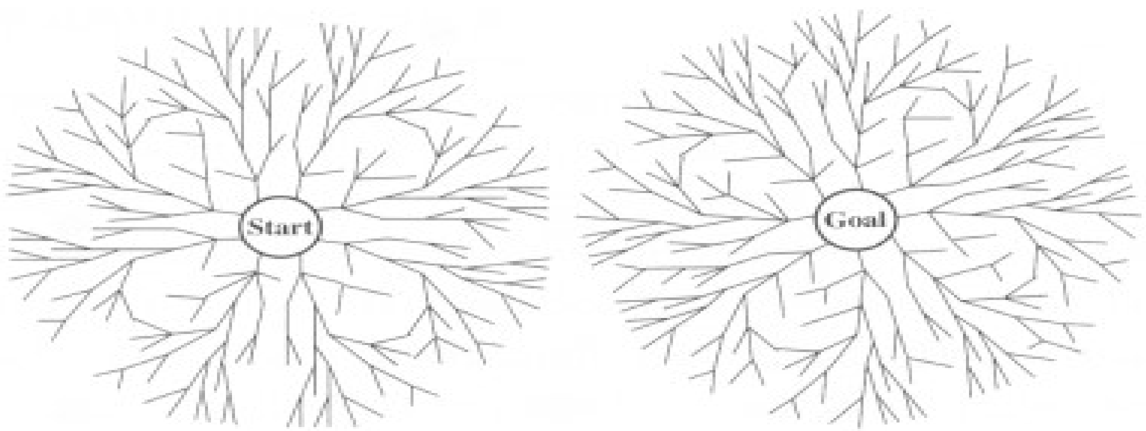
Bidirectional Search

- Run two simultaneous searches
  - One forward from the initial state another backward from the goal
  - Stop when the two searches meet
- However, computing backward is difficult
  - A huge amount of goal states
  - At the goal state, which actions are used to compute it?
  - Can the actions be reversible to computer its predecessors?

pra-sâmi

41

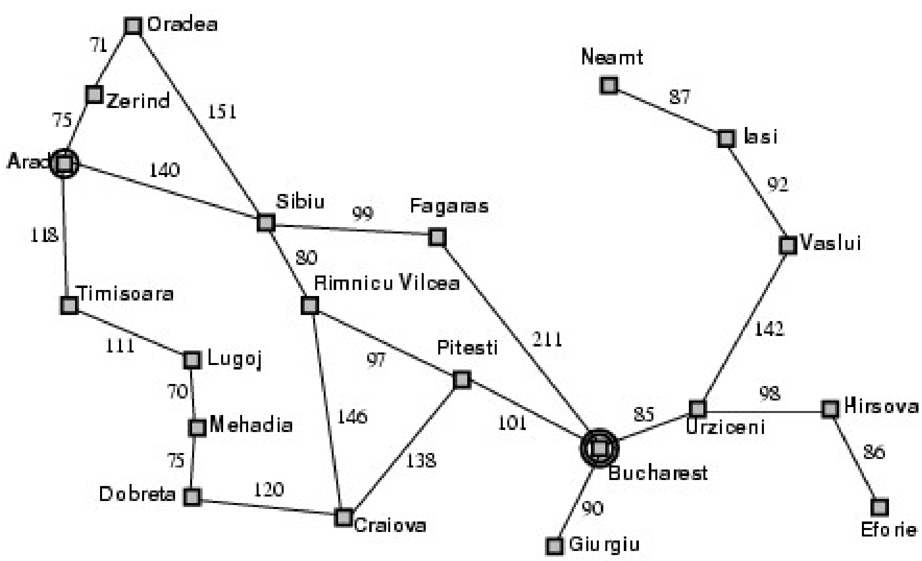
Bidirectional Search



pra-sâmi

42

Bidirectional Search

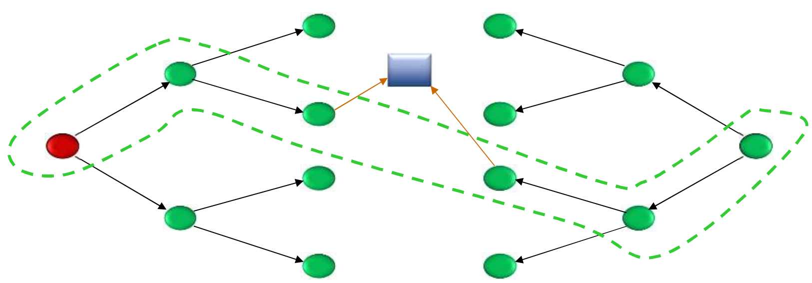


pra-sâmi

43

Bidirectional Strategy

2 fringe queues: FRINGE1 and FRINGE2



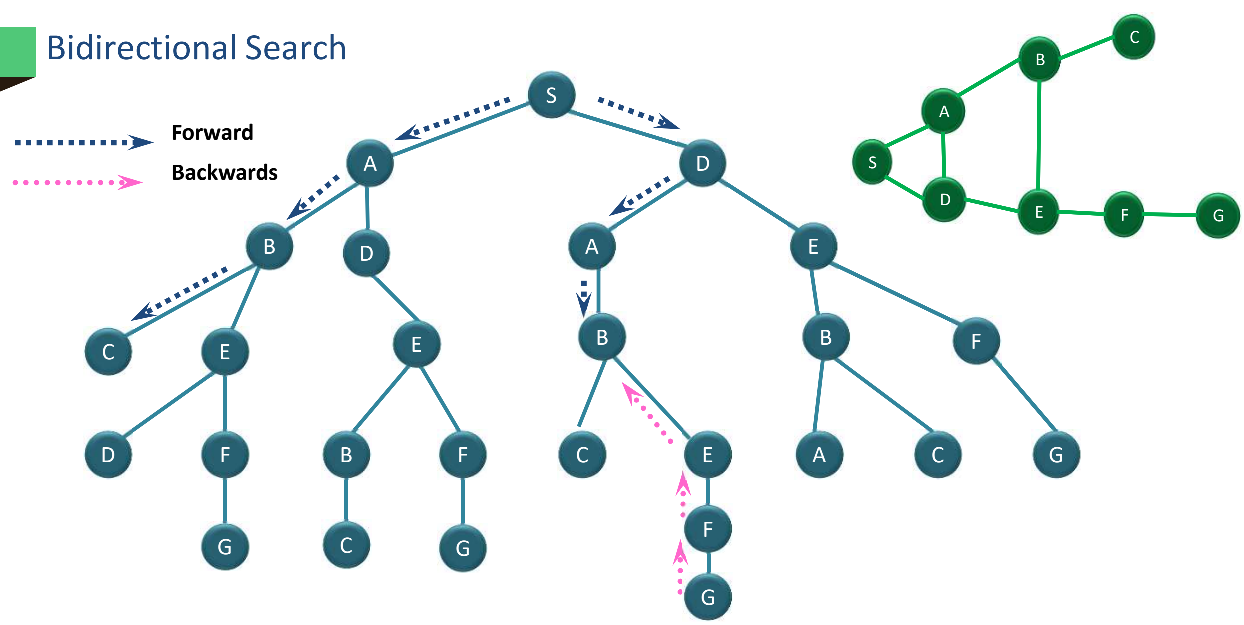
Time and space complexity = $O(b^{d/2}) \ll O(b^d)$

4/11/2024

pra-sâmi

44

Bidirectional Search



4/11/2024

pra-sâmi

45

Bidirectional Search is Challenging

- The reduction in time complexity makes bidirectional search attractive,
  - But how do we search backward? This is not as easy as it sounds.
- Let the predecessors of a state x be all those states that have x as a successor
  - Bidirectional search requires a method for computing predecessors
  - When all the actions in the state space are reversible, the predecessors of x are just its successors.
- What we mean by "the goal" in searching "backward from the goal."
  - For the 8-puzzle and for finding a route in Romania, there is just one goal state, so the backward search is very much like the forward search
- If there are several explicitly listed goal states
  - for example, the two dirt-free goal states in vacuum world then we can construct a new dummy goal state whose immediate predecessors are all the actual goal states
  - But if the goal is an abstract description, such as the goal that "no queen attacks another queen" in the n-queens problem, then bidirectional search is difficult to use

pra-sâmi

46

Comparing search strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^c	Yes	No	No	Yes^c	Yes^{c,d}

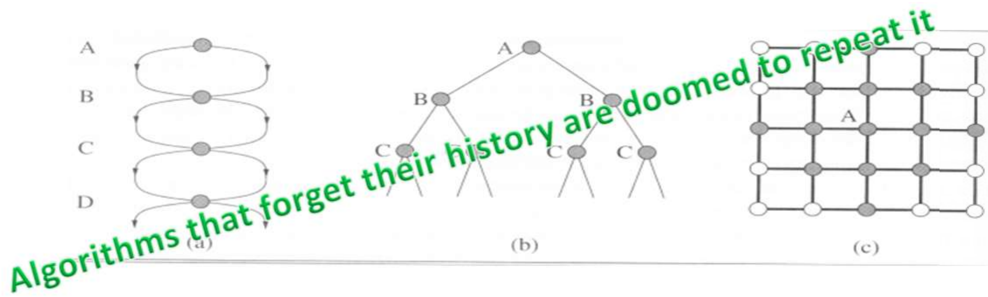
Figure 3.17 Evaluation of search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

pra-sâmi

47

Avoiding repeated states

- For all search strategies
  - There is possibility of expanding states that have already been encountered and expanded before, on some other path
  - May cause the path to be infinite → loop forever



pra-sâmi

48

Avoiding repeated states

- Try to remember all previously generated states
  - Space requirement goes up exponentially
  - Do not go back to parent in bidirectional graphs
  - May be do not go to ancestor
- Do not return to the state it just came from
  - Refuse generation of any successor same as its parent state
- Do not create paths with cycles
  - Refuse generation of any successor same as its ancestor states
- Do not generate any generated state
  - Not only its ancestor states, but also all other expanded states have to be checked against

pra-sâmi

49

Avoiding repeated states

- We then define a data structure
  - closed list:
  - a set storing every expanded node so far
  - If the current node matches a node on the closed list, discard it.

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

  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
      if EMPTY?(fringe) then return failure
      node ← REMOVE-FIRST(fringe)
      if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
      if STATE[node] is not in closed then
          add STATE[node] to closed
          fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```


50

Goal Not Found

51

Reflect…

- Abstraction is
  - the process to take out the irrelevant information
  - leave the most essential parts to the description of the states (Remove detail from representation)
  - Conclusion: Only the most important parts that are contributing to searching are used
- In Search tree : describe Initial State, Expanding, Leaf Node, Fringe Node
- Components of a Node are : State, Parent Node, Action, Path Cost, Depth
- Two different nodes can contain the same world state if that state is generated via two different search paths
- Heuristic search is also known as informed search

52

Next Session…

- Local Search Algorithms
- Hill-climbing
- Simulated Annealing
- Local Beam Search
- Genetic Algorithms
- Parallel Search

53

4/11/2024

pra-sâmi