Announcements

- Questions about Coding Homework 0?
- Coding Homework 1 will be released
  - Due 1/25 at 11:59pm
- Written Homework 1 will be released
  - Due 1/25 at 11:59pm

Uninformed Search

Forest Agostinelli
University of South Carolina

Topics Covered in This Class

- **Part 1: Search**
  - Pathfinding
    - Uninformed search
    - Informed search
  - Adversarial search
  - Optimization
    - Local search
    - Constraint satisfaction
- **Part 2: Knowledge Representation and Reasoning**
  - Propositional logic
  - First-order logic
  - Prolog
- **Part 3: Knowledge Representation and Reasoning Under Uncertainty**
  - Probability
  - Bayesian networks
- **Part 4: Machine Learning**
  - Supervised learning
    - Inductive logic programming
    - Linear models
    - Deep neural networks
    - PyTorch
  - Reinforcement learning
    - Markov decision processes
    - Dynamic programming
    - Model-free RL
  - Unsupervised learning
    - Clustering
    - Autoencoders

Outline

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

Motivation

- Donald Knuth conjectured that, starting with the number 4, a sequence of square root, floor, and factorial operations can create any desired positive integer
- How can we reach 5 from 4 using only these operations?

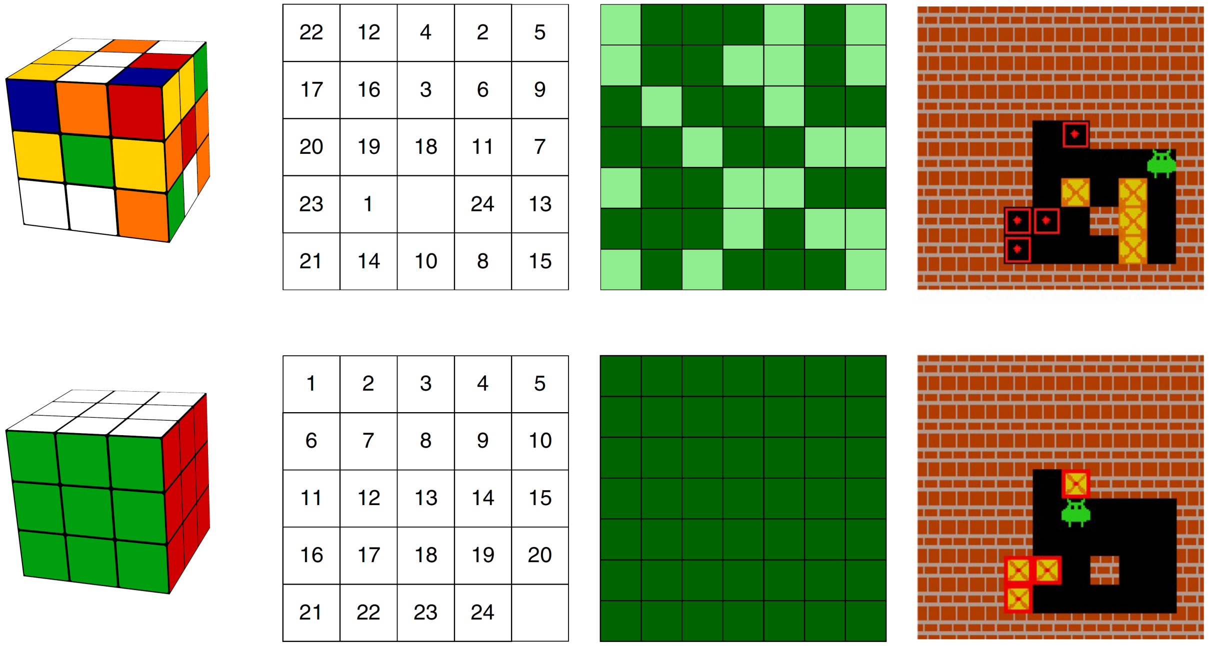
The Class of Problems

- For a pathfinding problem, we want to find a sequence of actions that transforms one state into a state that is a member of the set of goal states
  - Find a path to a goal state
- Pathfinding problems appear in many different areas, from robotics to the natural sciences
- Note: In general, while we may be able to pose a particular problem as a member of a class of problems (i.e. we can pose robotic object manipulation as a pathfinding problem). However, this does not mean pathfinding is the best way to solve the problem.
  - For example, machine learning, or some combination of the two, may perform better

$$S_0 \longrightarrow a_0 \longrightarrow S_1 \longrightarrow a_1 \longrightarrow S_1 \longrightarrow \quad \cdots \quad \longrightarrow S_g$$

Examples: Puzzles

- Puzzles often have an intuitive state (or set of states) that is considered the goal state
- Using pathfinding algorithms one can find solutions to these puzzles without knowing how to solve the puzzle
- Because of this, one can view AI as writing the algorithm to solve the puzzle for you
  - This theme will continue throughout this class and is one of the reasons AI is so powerful



Rubik's cube   24 puzzle   Lights Out (7×7)   Sokoban

Agostinelli, Forest, et al. "Solving the Rubik's cube with deep reinforcement learning and search." Nature Machine Intelligence 1.8 (2019): 356-363.

Examples: Theorem Proving and Chemical Synthesis

- Both theorem proving and chemical synthesis involve finding a path by combining building blocks (axioms or chemical compounds) to create a target structure (theorem to prove or chemical compound)

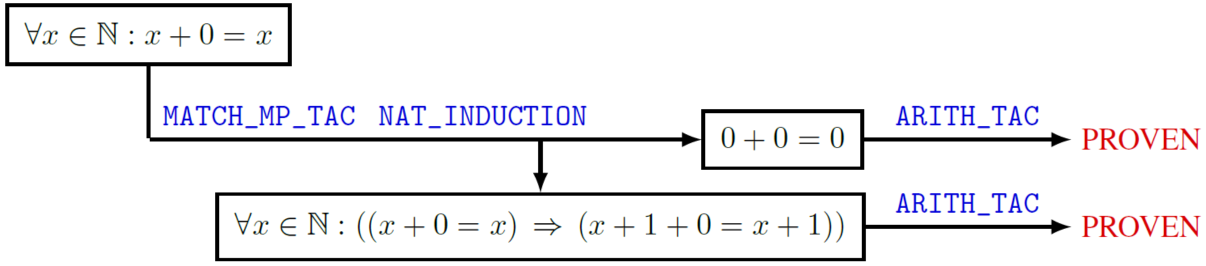
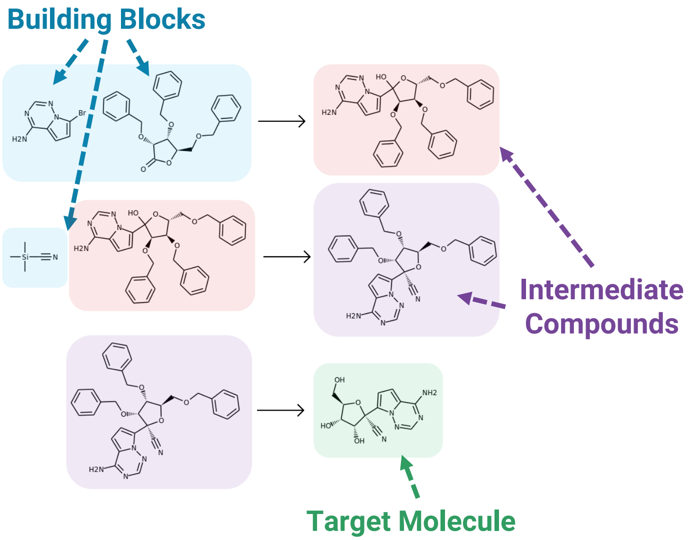


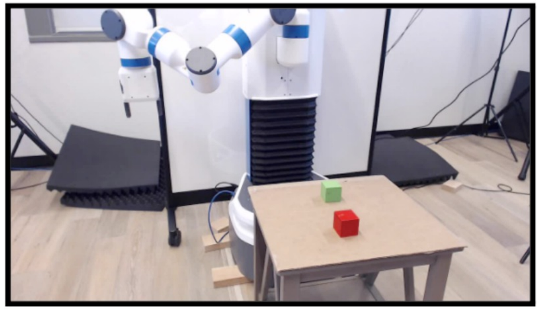
Figure 1: Formally proving $\forall x \in \mathbb{N} : x + 0 = x$.

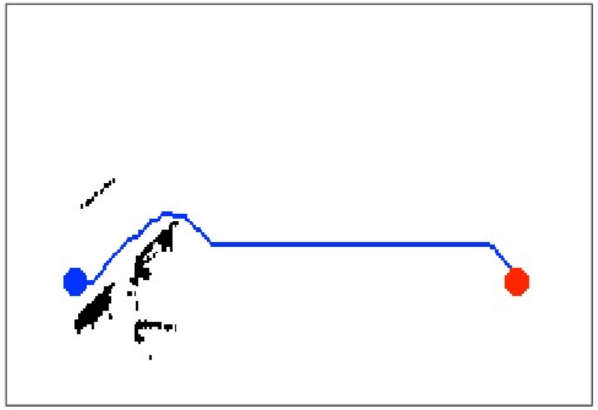


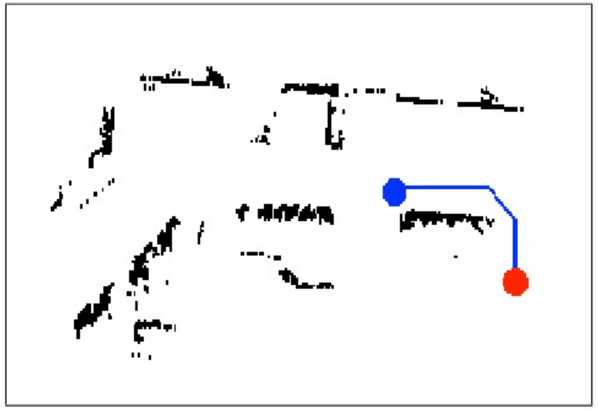
Chen, Binghong, et al. "Retro*: learning retrosynthetic planning with neural guided A* search." ICML, 2020.

Examples: Robotics

- Many problems in robotics are pathfinding problems.
- However, robots operate in a continuous environment, which poses a problem for many pathfinding algorithms
- Nonetheless, some techniques make use of discretization to use pathfinding algorithms







Andrychowicz, Marcin, et al. "Hindsight experience replay." NeurIPS (2017).
Likhachev, Maxim, et al. "Anytime Dynamic A*: An Anytime, Replanning Algorithm." *ICAPS*. Vol. 5. 2005.

Outline

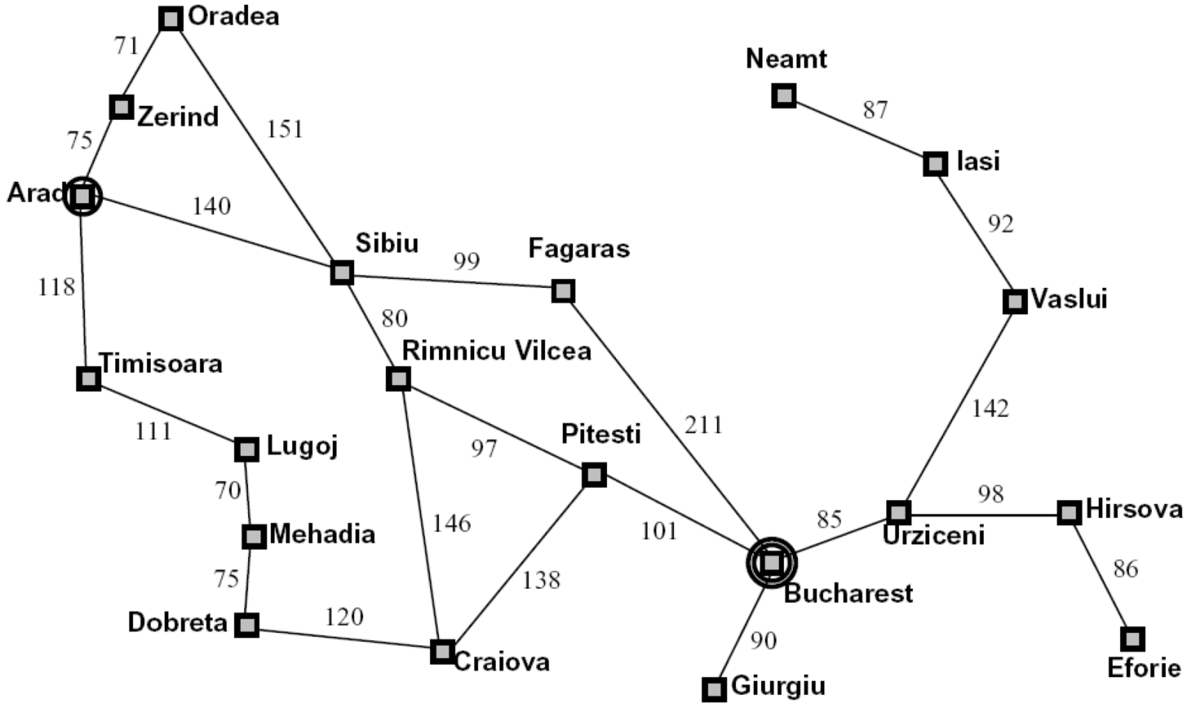
- Motivation
- Preliminaries
- Breadth-first search
- Uniform cost search
- Depth-first search
  - Depth-limited search
  - Iterative deepening search
- Bidirectional search

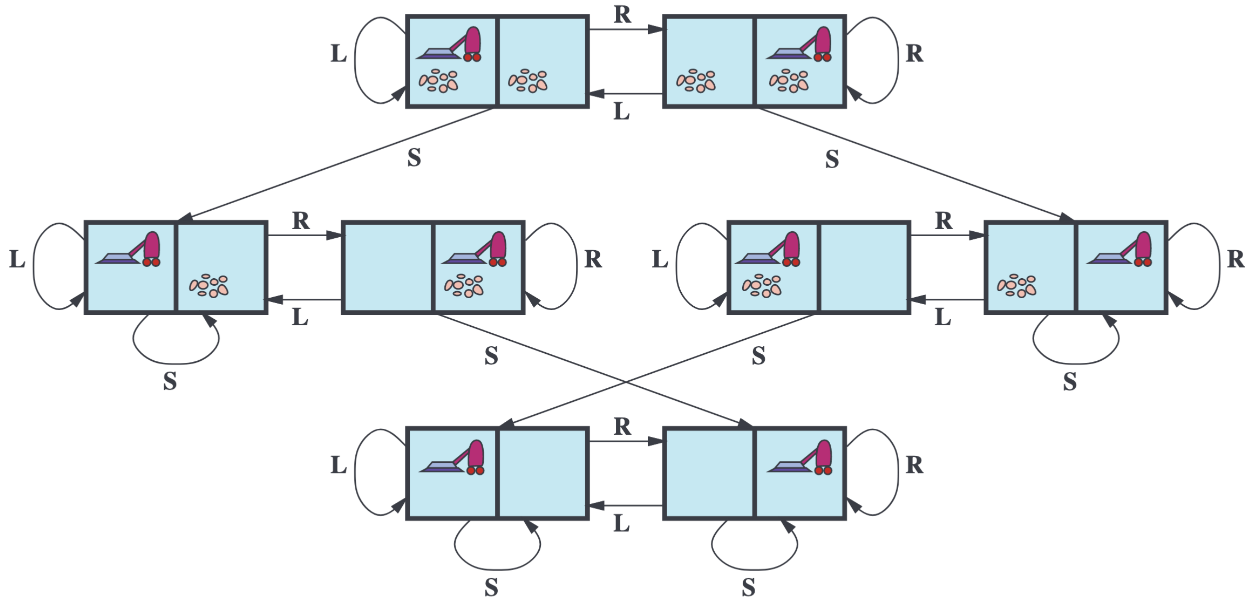
Defining a Pathfinding Problem

- **States $\mathcal{S}$**
  - Only keeps the details needed to solve the problem
- **Actions $\mathcal{A}$**
  - It is not always the case that every action can be taken in every state
- **Start state s_0**
- **Goal states $\mathcal{G} \subseteq \mathcal{S}$**
- **Transition model**
  - $s' = A(s, a)$
- **Transition cost function $c(s, a, s')$**
- Find a path from state s_0 to a state $s_g \in \mathcal{G}$
  - A minimum cost path is also referred to as an optimal or shortest path
  - There can be more than one optimal path

State Space Graph

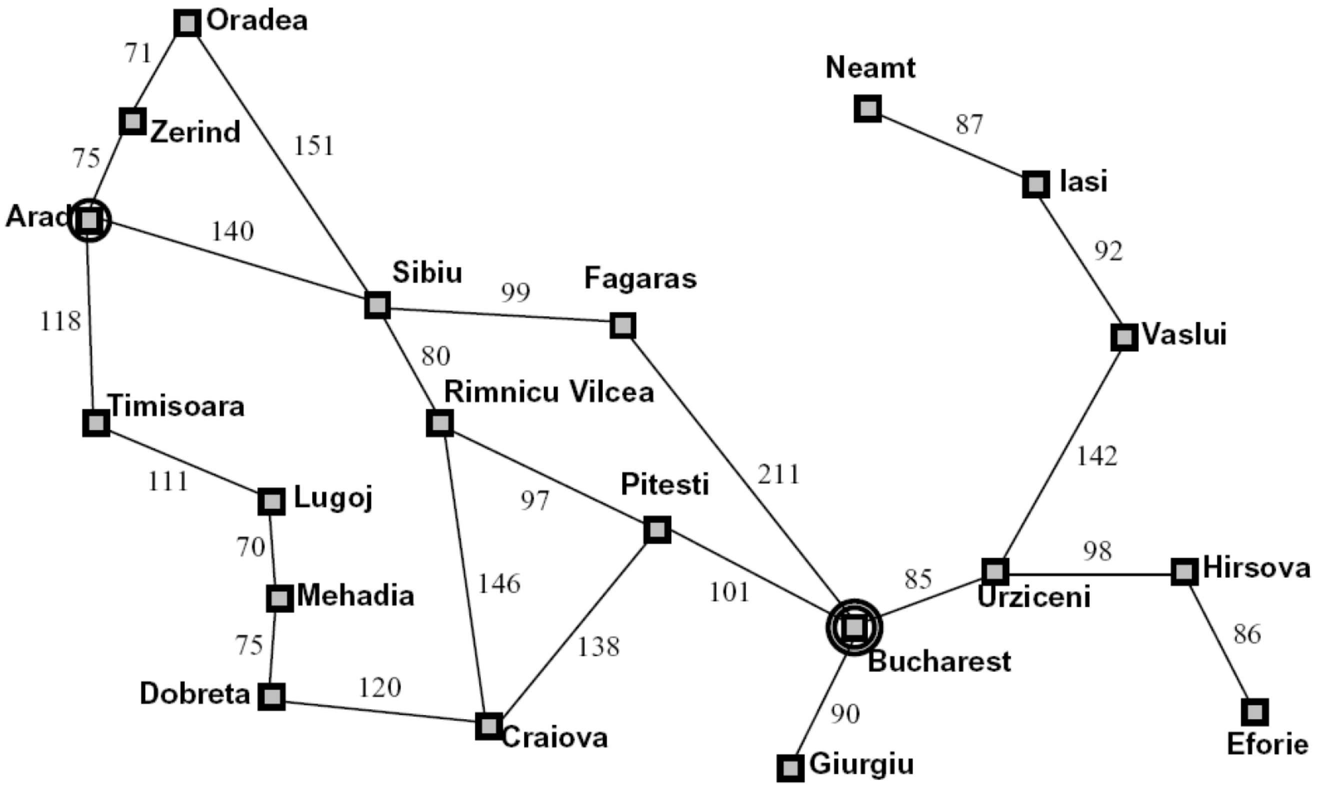
- Vertices: States
- Directed Edges: Actions
- Each state appears only once
- Pathfinding algorithms can be seen as finding a path between nodes in a graph





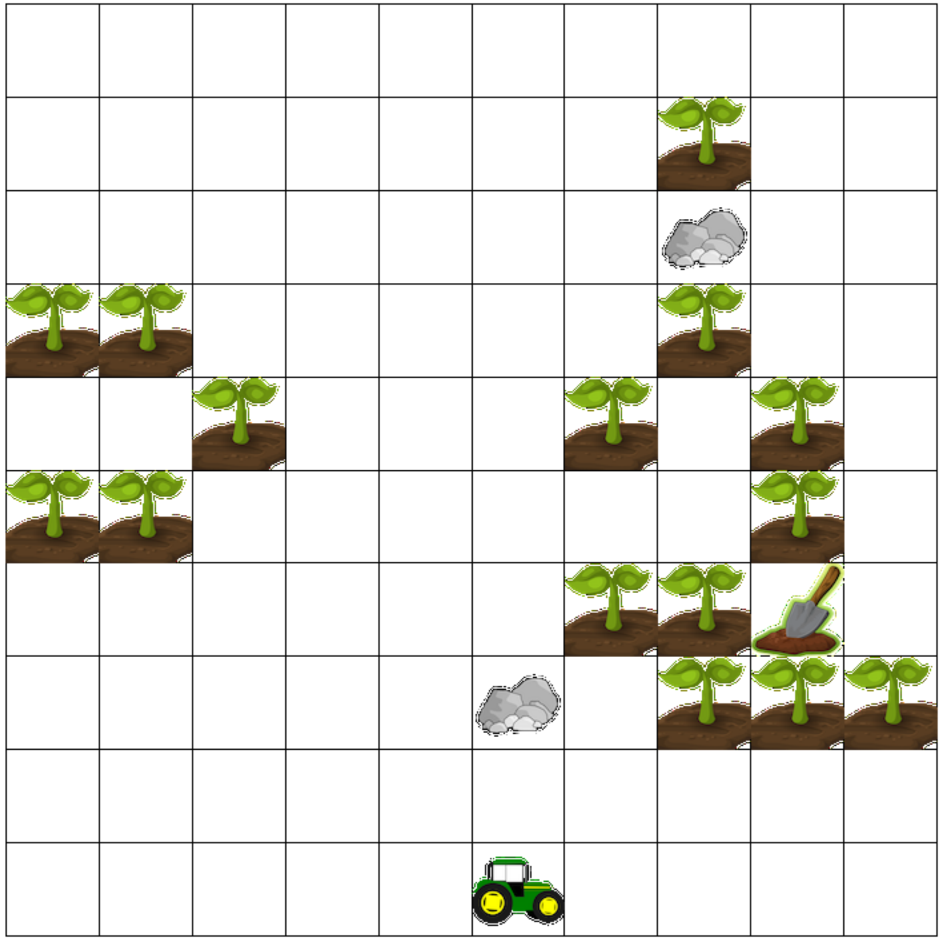
Example: Traveling in Romania

- Travel from Arad to Bucharest
- **States**
  - Cities
- **Actions**
  - Go to an adjacent city
- **Start state**
  - Arad
- **Goal state(s)**
  - Bucharest
- **Transition model**
  - Go to selected city
- **Transition cost function**
  - Driving time



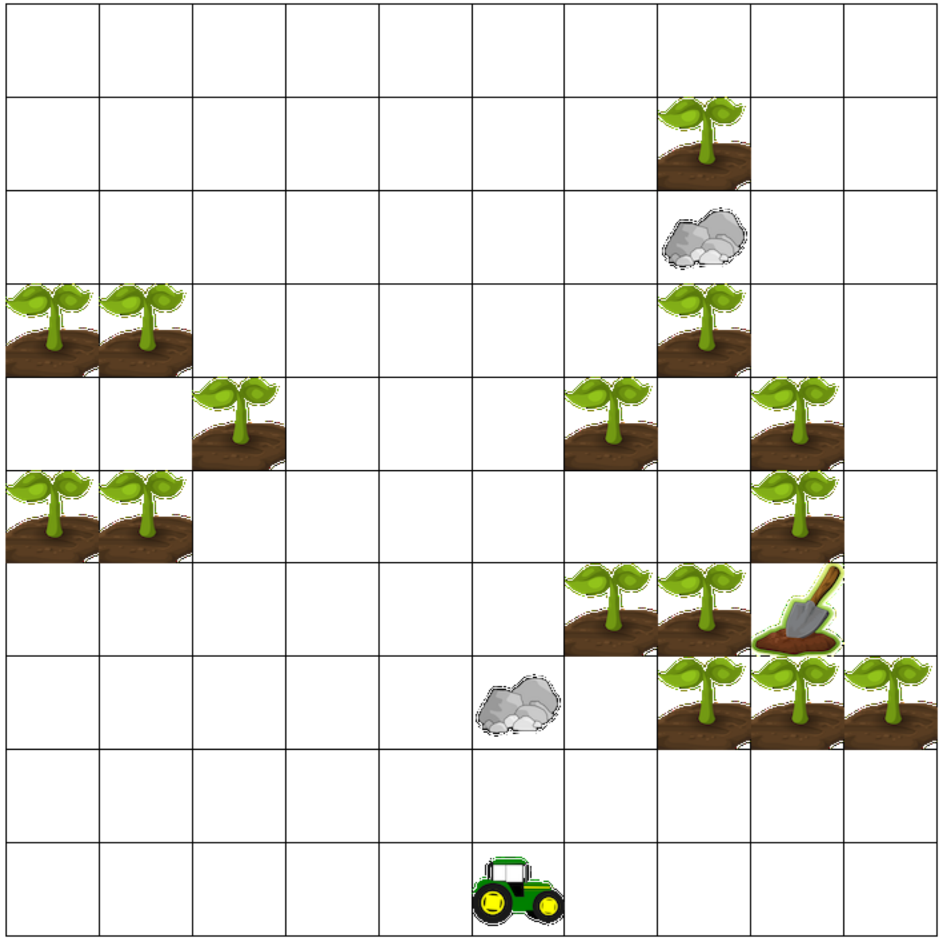
Example: AI Farm

- Move the tractor to the shovel
- **States**
  - Locations of tractor
  - 100 possible states
- **Actions**
  - Up, down, left, right
- **Start state**
  - Tractor's current location
- **Goal state(s)**
  - Tractor on the shovel's location
- **Transition model**
  - Tractor moves in direction of action
  - Stays in place if it hits a wall
- **Transition cost function**
  - 1 for each step
  - 10 for driving on a rock
  - 50 for driving on a plant



Example: AI Farm

- Water all the plants
- **States**
  - 15 plants, can be watered or not watered 2^{15}
  - 100 possible locations for tractor
  - $100 * 2^{15} \approx 3.27 \times 10^6$
- **Actions**
  - Up, down, left, right, water up, water down, water left, water right
- **Start state**
  - Tractor's current location and status of plants (watered/un-watered)
- **Goal state(s)**
  - All plants watered (100 goal states)
- **Transition model**
  - Tractor moves/waters in direction of action
  - Stays in place if it hits a wall
  - If plant was un-watered, it changes to watered
- **Transition cost function**
  - 1 for each step
  - 10 for driving on a rock
  - 50 for driving on a plant



Pathfinding Algorithms

- Expand nodes according to some priority until a goal node is selected for expansion
- Use a priority queue to sort nodes according to priority
  - This is referred to as OPEN or the “fringe”
  - For some algorithms, it can be implemented as a simple FIFO or LIFO queue
- Some algorithms use a CLOSED set to remember the nodes that have been generated
  - Sometimes referred to as “reached”
  - Prevents redundant node expansions

Nodes

- **Node:** Bookkeeping data structure for search
  - State
  - Parent node
  - Action
    - Action that the parent took to generate this node
  - Path cost
    - Cost of path from the start node to current node
- There can be multiple nodes with the same state
- We will refer to a node with the start state as n_0 and with a goal state as n_g
- A node is expanded when we use the transition function to generate all its children

Node Expansion

- Apply every possible action to the state associated with the node

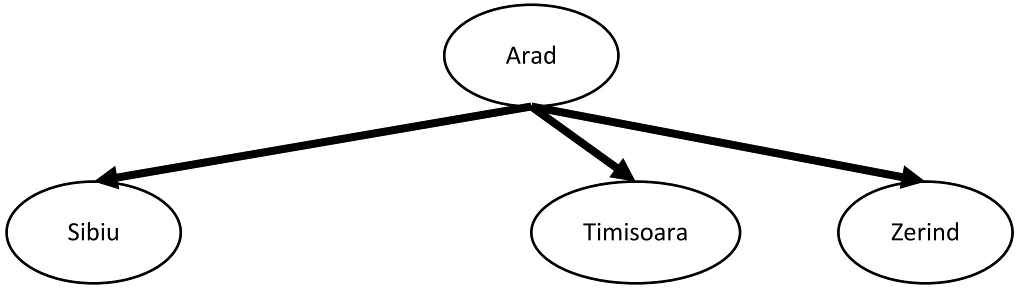
for each action a for $n.s$

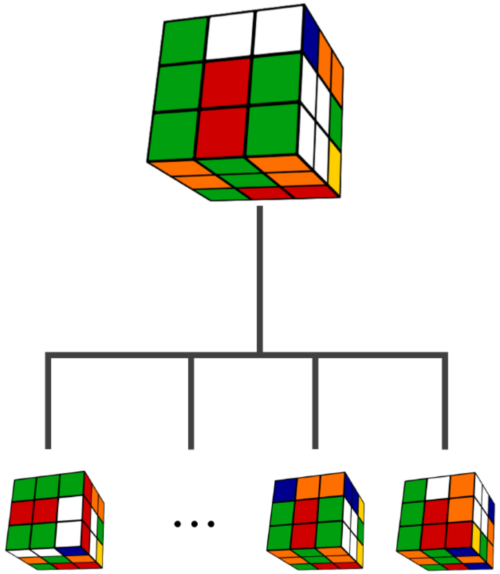
$s' = A(n.s, a)$ // next state

$g = n.g + c(n.s, a, s')$ // path cost

$d = n.d + 1$ //depth

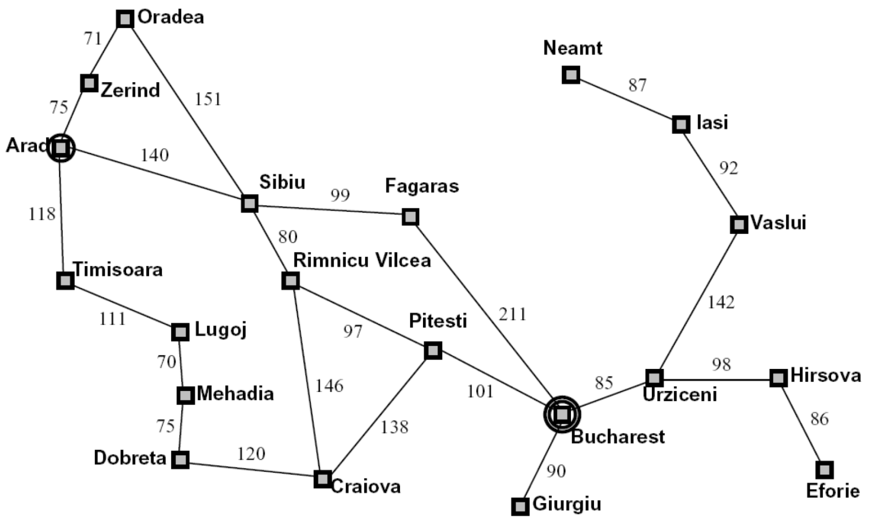
$n_c = Node(s', n, a, g, d)$ //new node



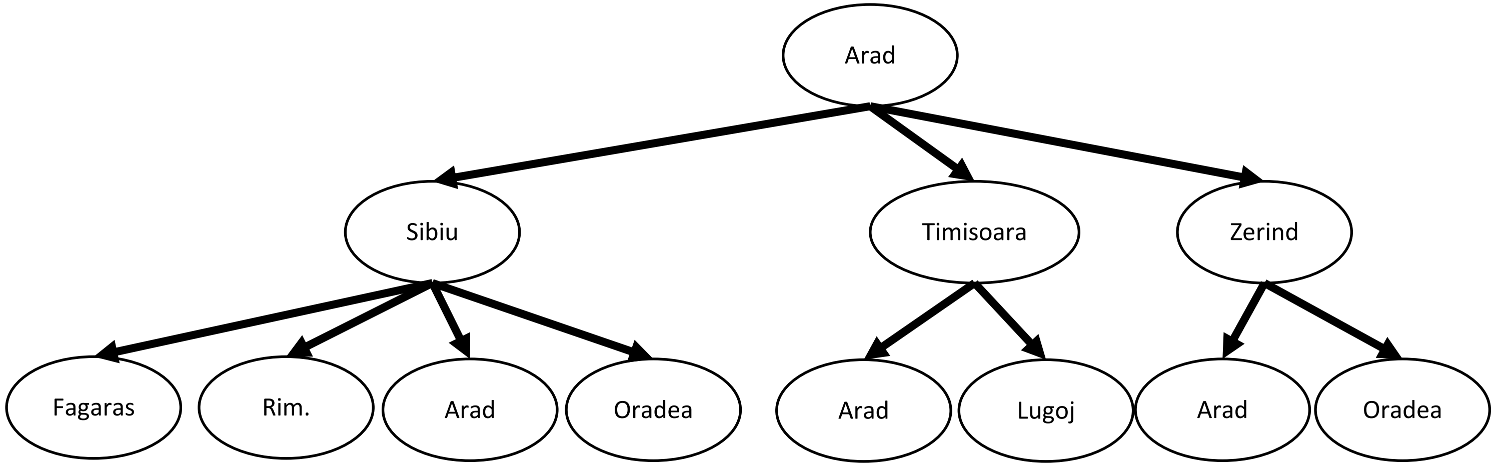


Search Tree

- Pathfinding algorithms can form a tree where states appear multiple times; representing different paths one can take to the same state
  - Remember, every node except for the root node has exactly one parent
- Vertices: States
- Directed Edges: Actions



State space graph



Search tree

How to Analyze Search Algorithms

- Completeness
  - Is complete if an only if it always finds a solution (if a solution exists)
- Time complexity
  - Number of nodes generated
- Space complexity
  - Maximum number of nodes in memory
- Optimality
  - Is optimal if and only if it always finds a least-cost solution

How to Analyze Search Algorithms

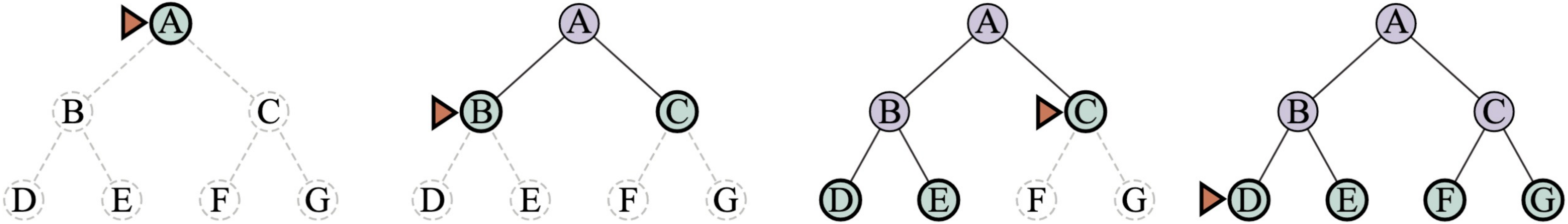
Criterion	Breadth-First Search	Uniform Cost Search	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional Search
Complete?						
Time						
Space						
Optimal						

Outline

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

Breadth-First Search

- Prioritize the shallowest nodes
- For breadth-first search, we do not have to wait until the goal node is selected for expansion, we can terminate when the goal state is generated



Node to be expanded next

Not yet generated

In OPEN

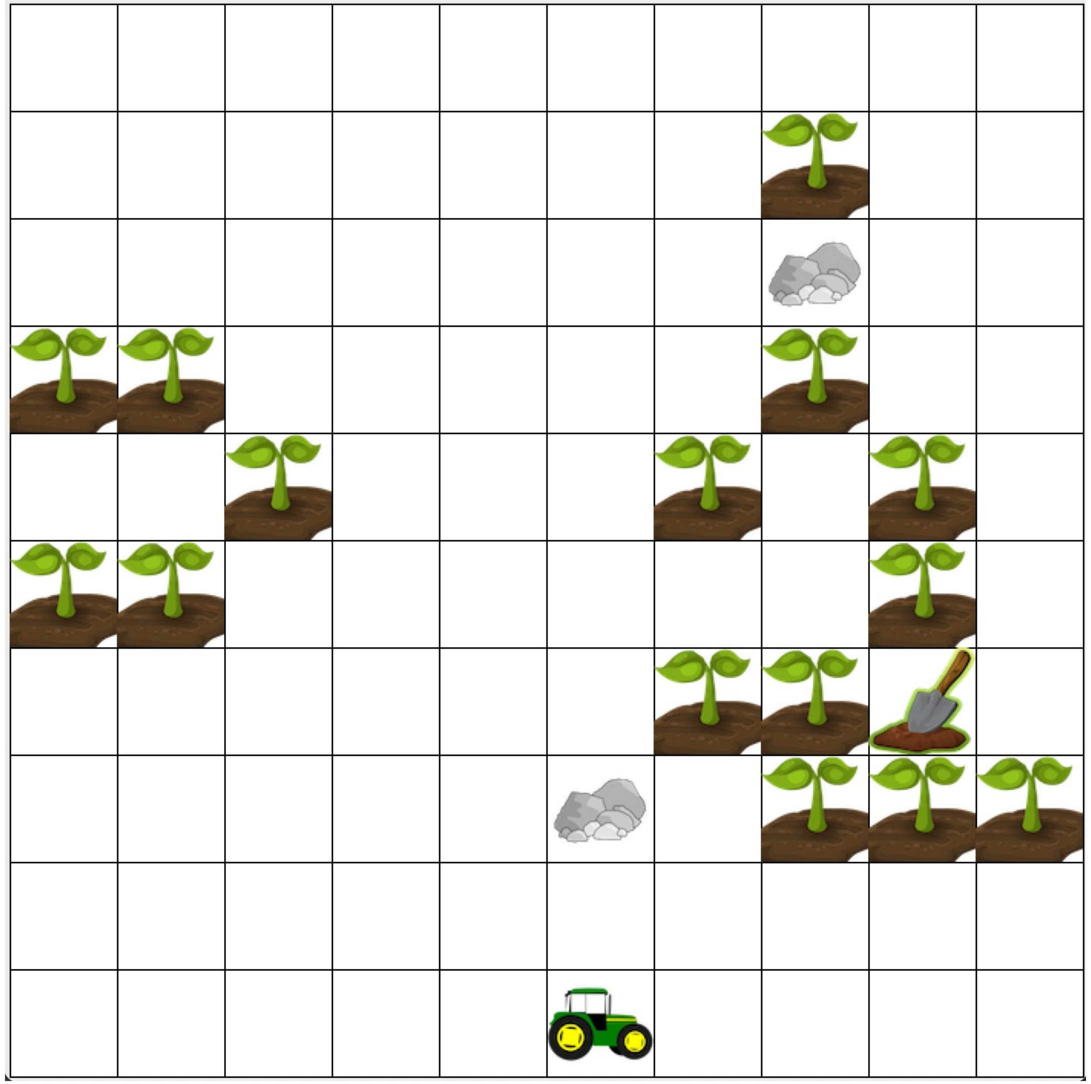
Expanded

Breadth First Search

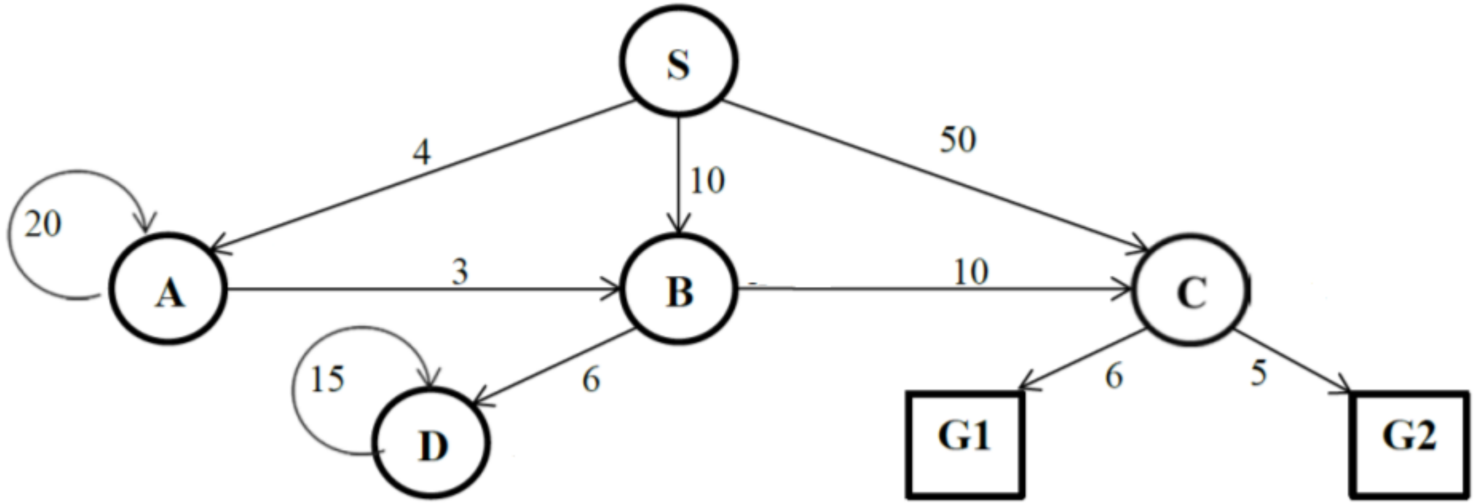
function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution node or *failure*
  node ← NODE(*problem*.INITIAL)
  if *problem*.IS-GOAL(*node*.STATE) **then return** *node*
  frontier ← a FIFO queue, with *node* as an element
  reached ← {*problem*.INITIAL}
  while not IS-EMPTY(*frontier*) **do**
    node ← POP(*frontier*)
    for each *child* **in** EXPAND(*problem*, *node*) **do**
      s ← *child*.STATE
      if *problem*.IS-GOAL(*s*) **then return** *child*
      if *s* is not in *reached* **then**
        add *s* to *reached*
        add *child* to *frontier*
  return *failure*

Breadth-first search is a special case where we can do the goal test when nodes are generated instead of when they are selected for expansion

Breadth First Search: AI Farm

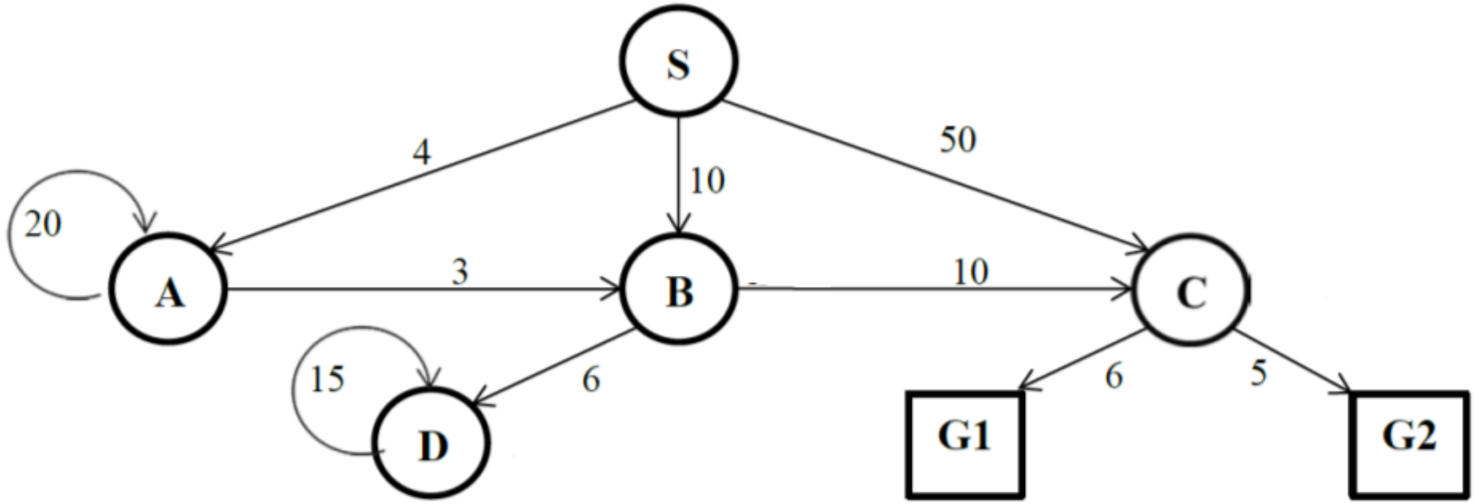


Solve: Breadth-First Search



- Order of expansion?
- Path found?

Solve: Breadth-First Search



- Order of expansion: S, A, B, C, (G1)
- Path found: S, C, G1

Breadth-First Search

Criterion	Breadth-First Search	Uniform Cost Search	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional Search
Complete?	Yes[a]					
Time	$O(b^d)$					
Space	$O(b^d)$					
Optimal	Yes[c]					

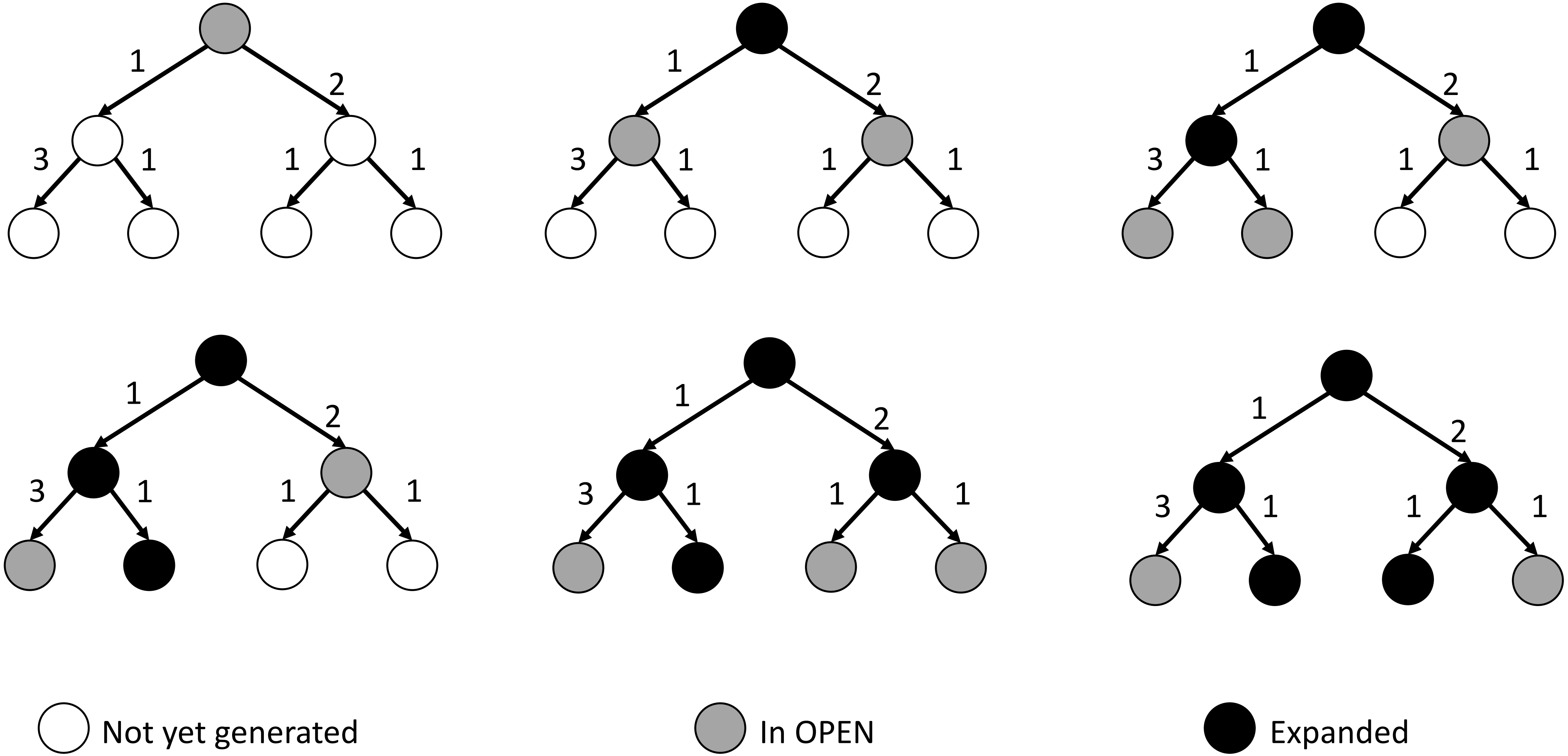
Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c]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

Outline

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

Uniform Cost Search (Dijkstra's algorithm)

- Prioritize nodes with the lowest path cost (ties broken arbitrarily)



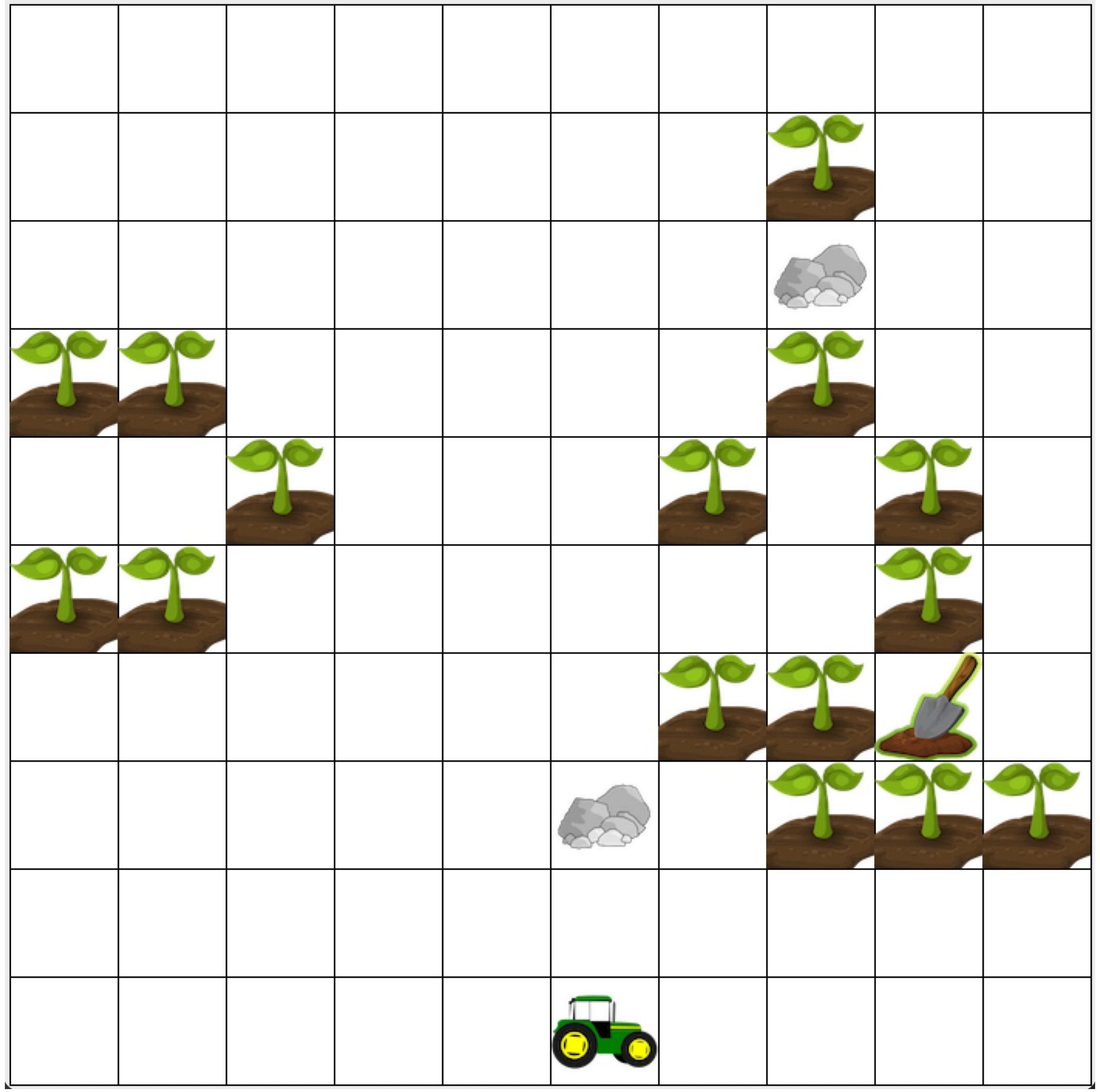
Uniform Cost Search

- Uniform cost search is best first search where the function f returns the path cost

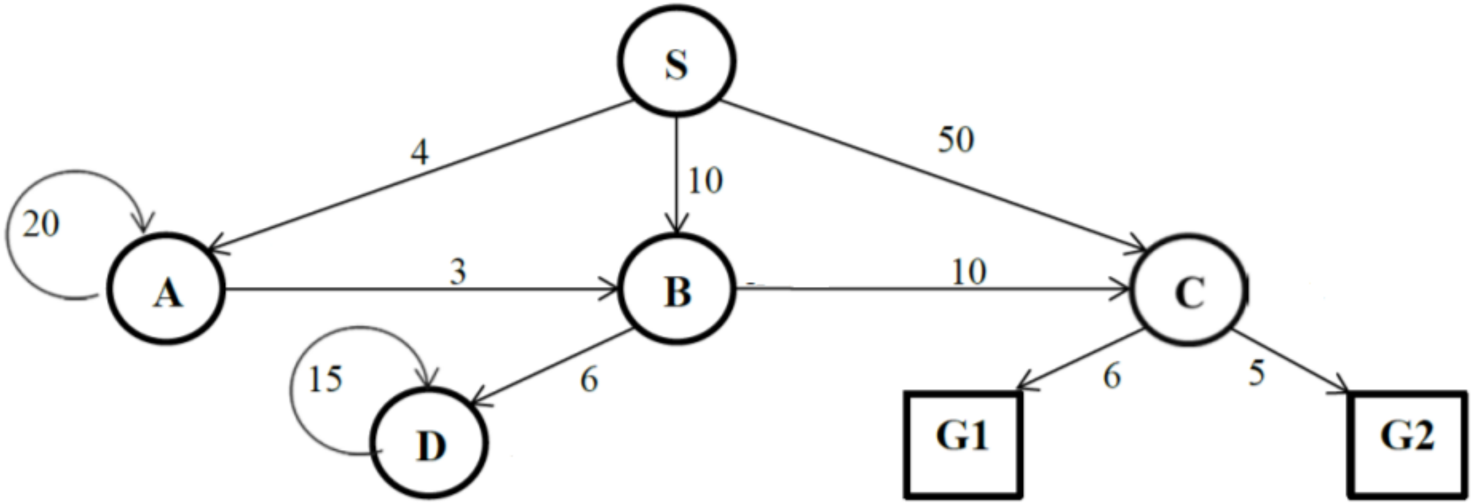
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
    node ← NODE(STATE=problem.INITIAL)
    frontier ← a priority queue ordered by f, with node as an element
    reached ← a lookup table, with one entry with key problem.INITIAL and value node
    while not IS-EMPTY(frontier) do
        node ← POP(frontier)
        if problem.IS-GOAL(node.STATE) then return node
        for each child in EXPAND(problem, node) do
            s ← child.STATE
            if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
                reached[s] ← child
                add child to frontier
    return failure
```

Only do a goal test when when the node is selected for expansion!

Uniform Cost Search: AI Farm

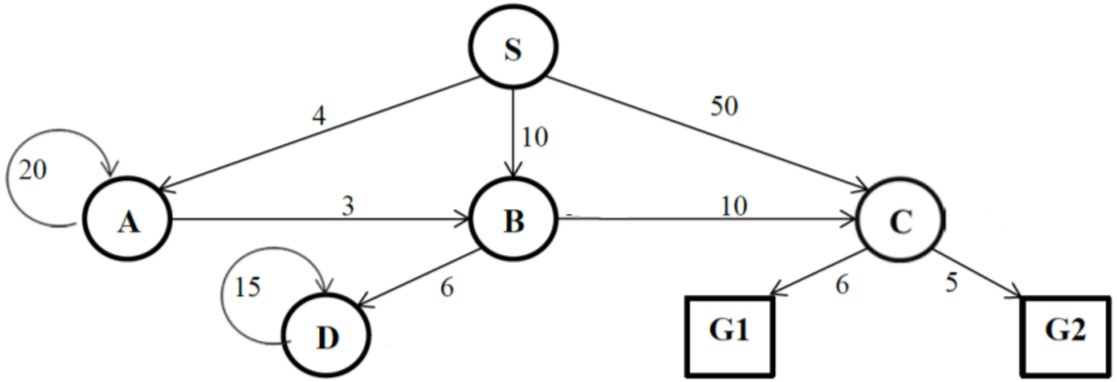


Solve: Uniform Cost Search



- Order of expansion?
- Path found?

Solve: Uniform Cost Search



- Order of expansion: S, A, B, B, D, C, (G2)
- Path found: S, A, B, C, G2

CLOSED	OPEN	Expanded
S: 0	S: 0	S
S: 0, A: 4, B: 10, C: 50	A: 4, B: 10, C: 50	A
S: 0, A: 4, B: 7, C: 50	B: 7, B: 10, C: 50	B
S: 0, A: 4, B: 7, C: 17, D: 13	B: 10, D: 13, C: 17, C: 50	B
S: 0, A: 4, B: 7, C: 17, D: 13	D: 13, C: 17, C: 50	D
S: 0, A: 4, B: 7, C: 17, D: 13	C: 17, C: 50	C
S: 0, A: 4, B: 7, C: 17, D: 13, G1: 23, G2: 22	G2: 22, G1: 23, C: 50	(G2)

Uniform Cost Search

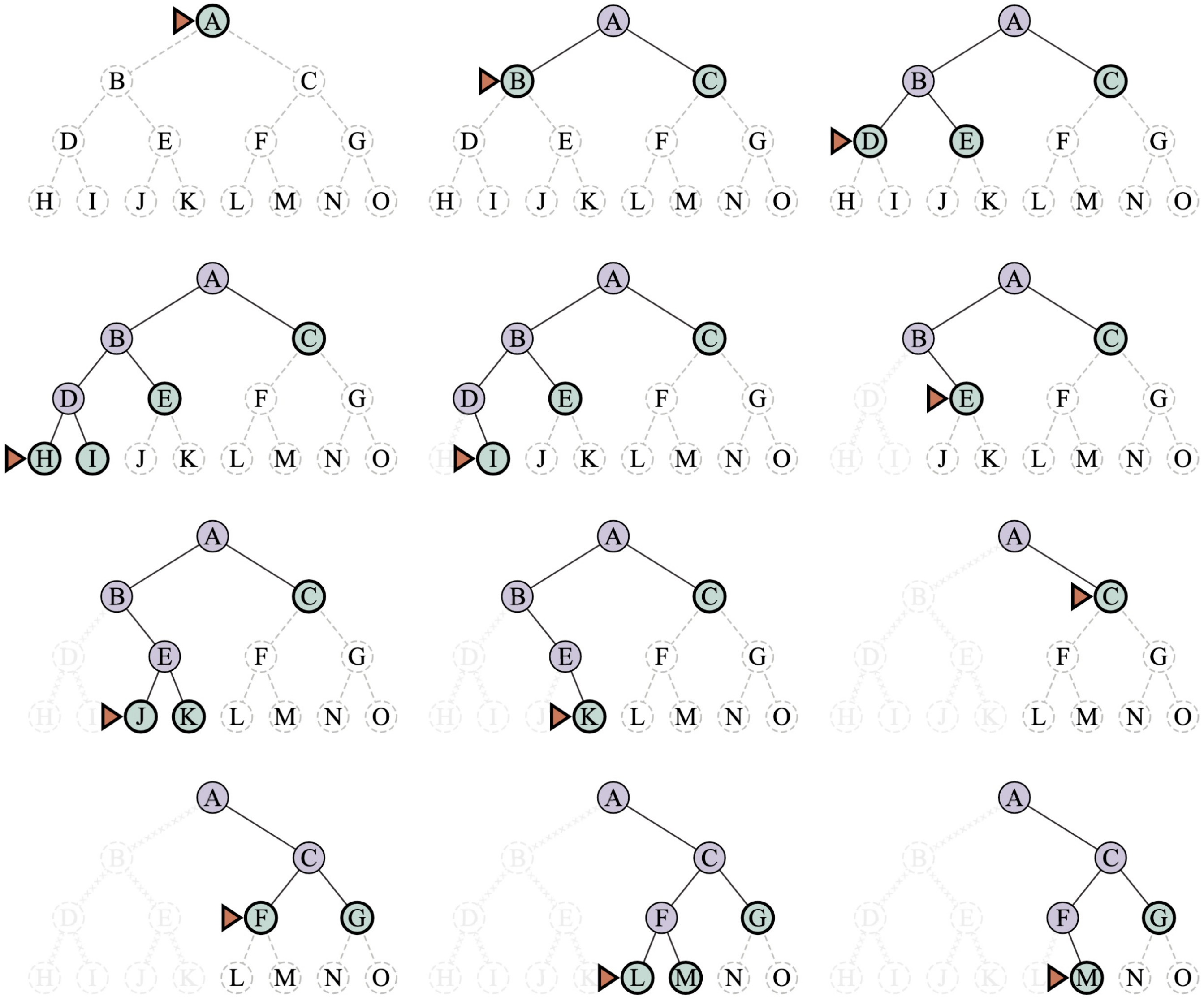
Criterion	Breadth-First Search	Uniform Cost Search	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional Search
Complete?	Yes[a]	Yes[a,b]				
Time	$O(b^d)$	$O(b^{1+floort(C^*/\varepsilon)})$				
Space	$O(b^d)$	$O(b^{1+floort(C^*/\varepsilon)})$				
Optimal	Yes[c]	Yes				

Evaluation of tree-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.

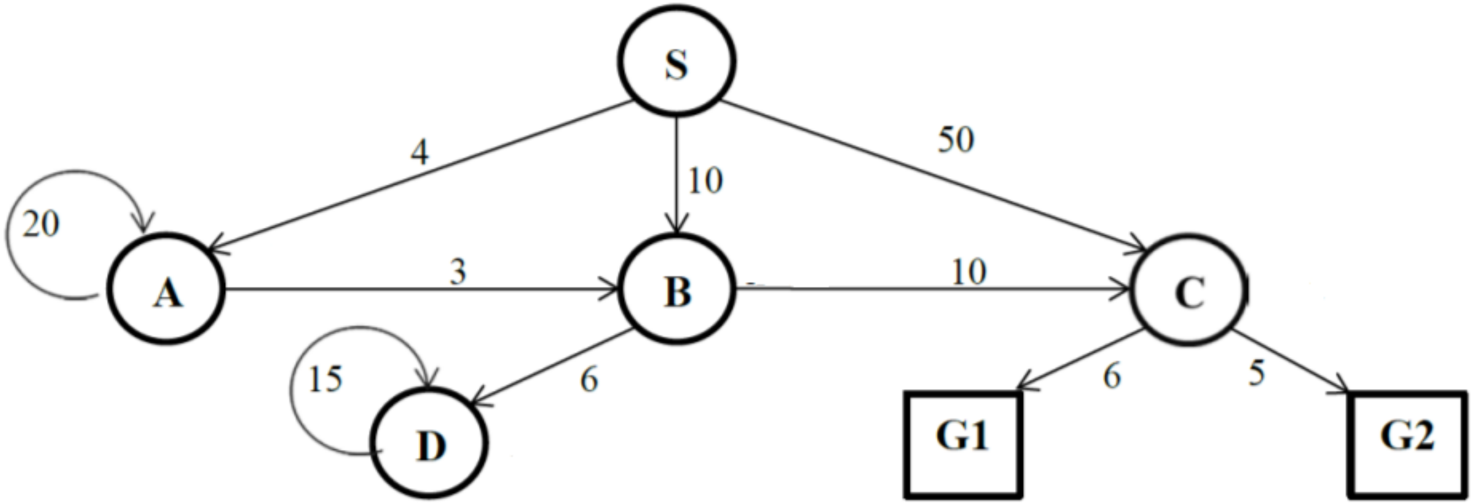
Outline

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

Depth-First Search



Depth-First Search



- Depending on the order of nodes returned during expansion, we could have an infinite loop at A

Depth-First Search

Criterion	Breadth-First Search	Uniform Cost Search	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional Search
Complete?	Yes[a]	Yes[a,b]	No			
Time	$O(b^d)$	$O(b^{1+floort(C^*/\varepsilon)})$	$O(b^m)$			
Space	$O(b^d)$	$O(b^{1+floort(C^*/\varepsilon)})$	$O(bm)$			
Optimal	Yes[c]	Yes	No			

Evaluation of tree-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.

Depth-Limited Search

```
function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

Typo in the book. > should be changed to ≥

Depth-limited Search

Criterion	Breadth-First Search	Uniform Cost Search	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional Search
Complete?	Yes[a]	Yes[a,b]	No	No		
Time	$O(b^d)$	$O(b^{1+floort(C^*/\varepsilon)})$	$O(b^m)$	$O(b^l)$		
Space	$O(b^d)$	$O(b^{1+floort(C^*/\varepsilon)})$	$O(bm)$	$O(bl)$		
Optimal	Yes[c]	Yes	No	No		

Evaluation of tree-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.

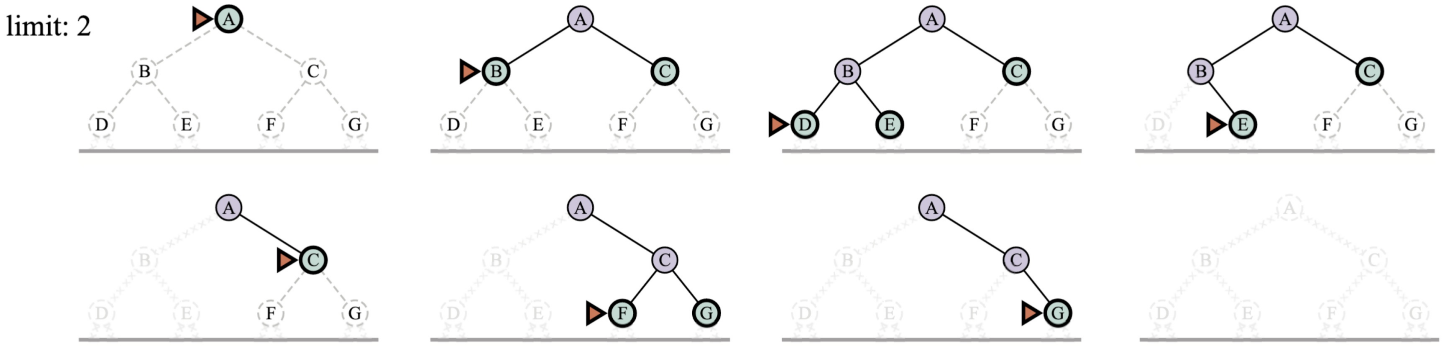
Iterative Deepening Search

- Do depth-limited starting with a limit of 0
- Increase limit until a solution or a failure is encountered

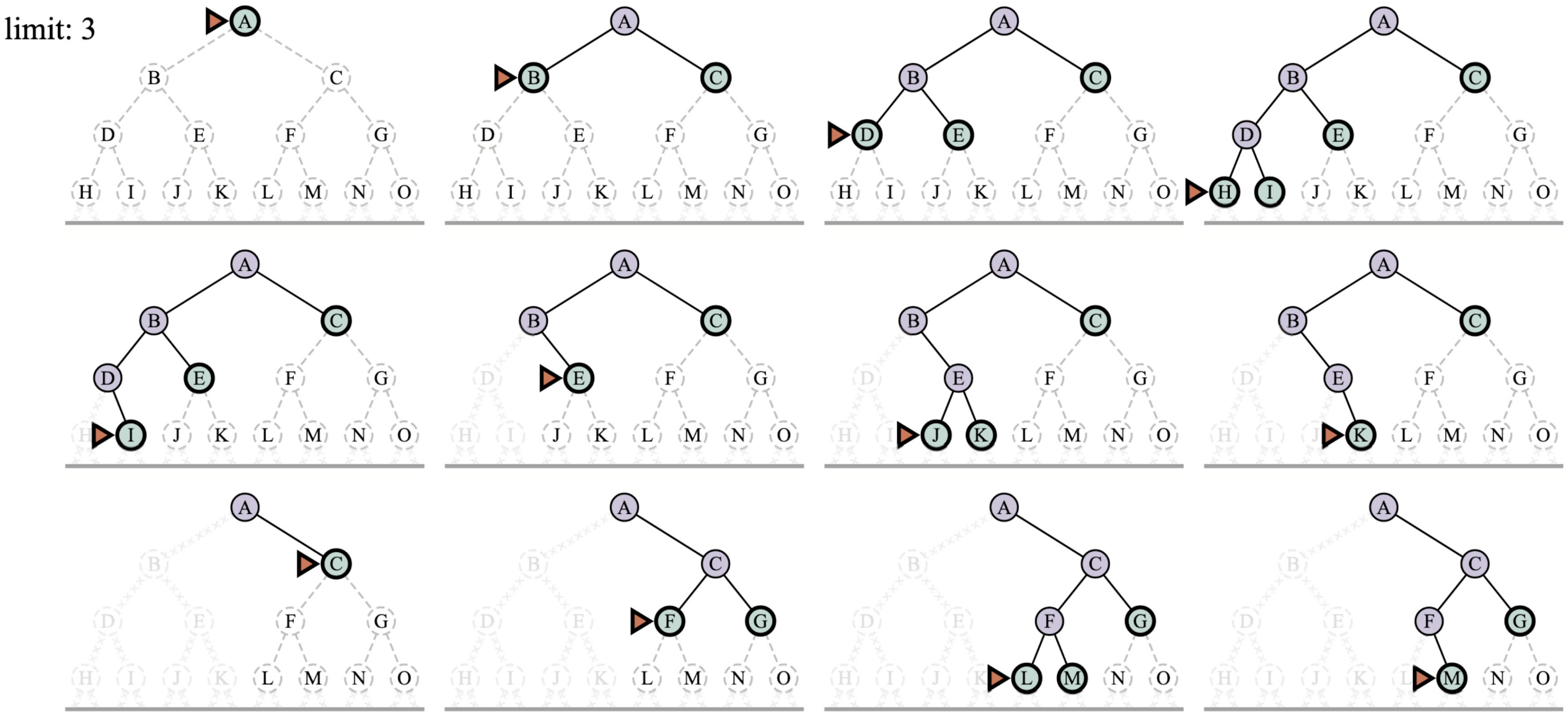
Iterative Deepening Search

limit: 1

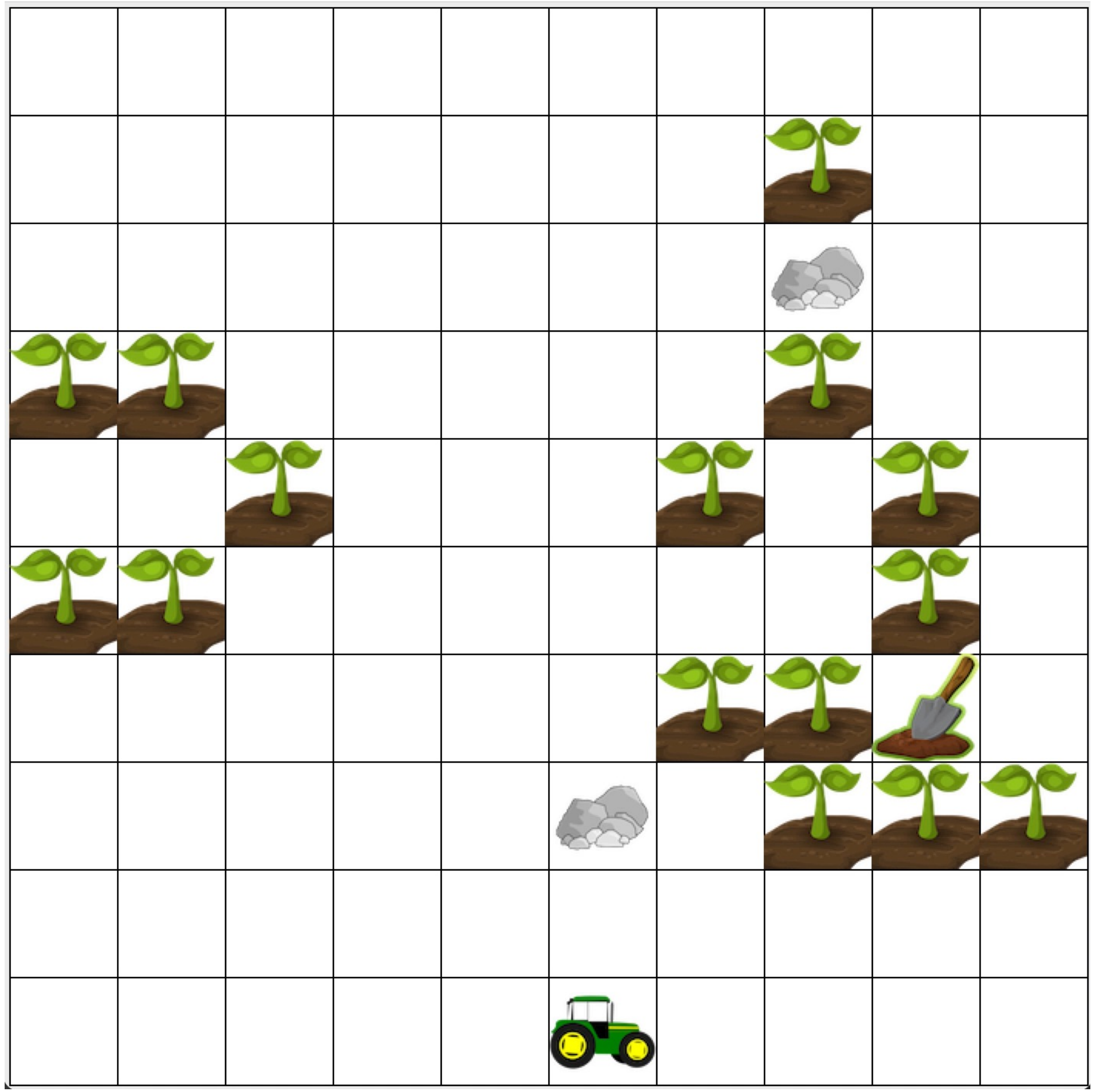
Iterative Deepening Search



Iterative Deepening Search



Iterative Deepening Search: AI Farm



Iterative Deepening Search

Criterion	Breadth-First Search	Uniform Cost Search	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional Search
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	
Time	$O(b^d)$	$O(b^{1+floort(C^*/\varepsilon)})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	
Space	$O(b^d)$	$O(b^{1+floort(C^*/\varepsilon)})$	$O(bm)$	$O(bl)$	$O(bd)$	
Optimal	Yes[c]	Yes	No	No	Yes[c]	

Evaluation of tree-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.

Outline

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

Bidirectional Search

- Does breadth-first search in both directions
- Must be able to implement a reverse transition function
- Terminates when the OPEN queues (frontiers) intersect

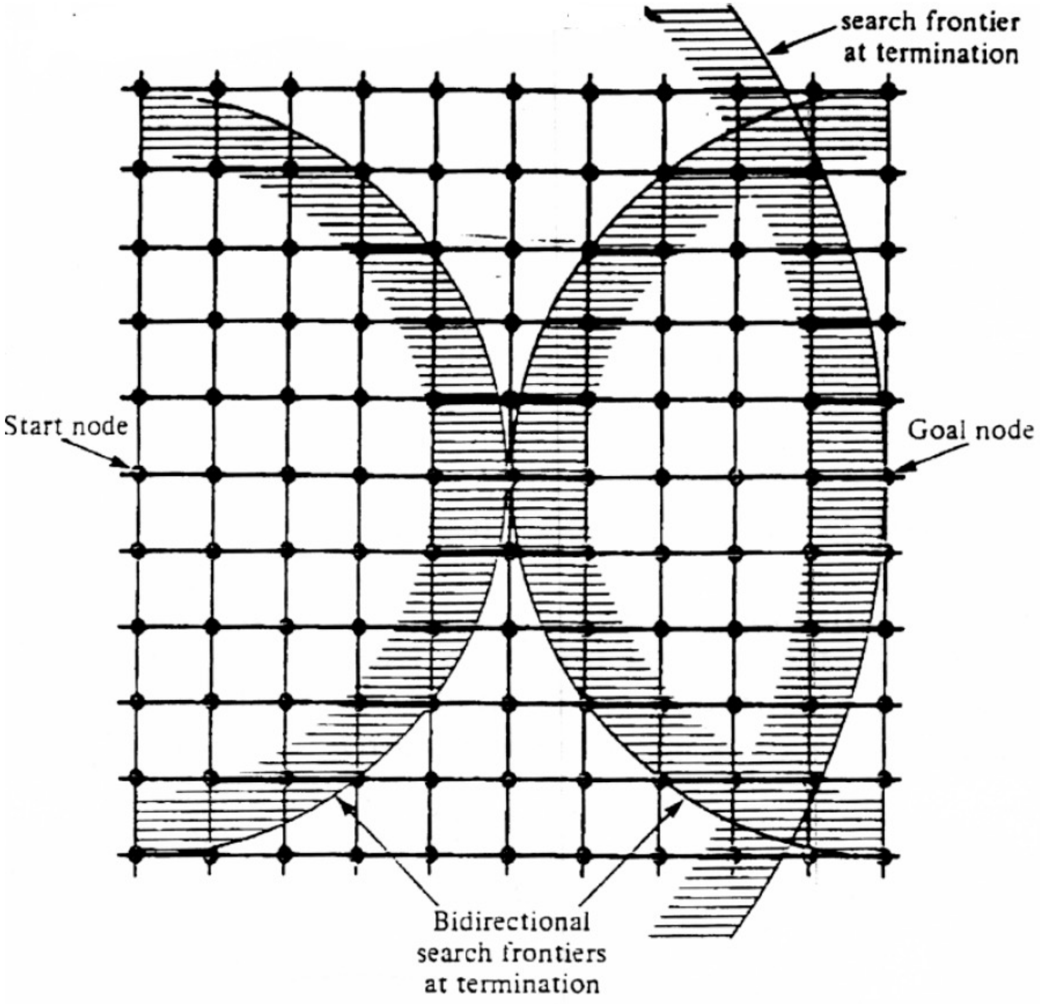


Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

Bidirectional Search

Criterion	Breadth-First Search	Uniform Cost Search	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional Search
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+floort(C^*/\varepsilon)})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+floort(C^*/\varepsilon)})$	O(bm)	O(bl)	O(bd)	$O(b^{d/2})$
Optimal	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Evaluation of tree-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.

Summary

- Breadth-first search
  - Prioritizes shallowest
  - FIFO queue
- Uniform-cost search
  - Prioritizes the ones with the lowest path cost
  - Priority queue ordered by path cost
- Depth-first search (and variants)
  - Prioritizes deepest
  - LIFO queue (stack)
- Bidirectional search
  - Breadth-first search from the start and goal

Next Time

- While uniform cost search is guaranteed to find a shortest path, it expanded almost every node in the state space
- Next time, we will look at informed search strategies that estimate how close each node is to solving the problem to better prioritize node expansion