COSC 4368 Fundamentals of Artificial Intelligence

Lecture 3: Search 2

August 28th, 2023

Overview

- Last time:
  - Framework for AI techniques: modeling, inference, learning
  - Problem examples investigated by subfields of AI
  - Brief overview of search

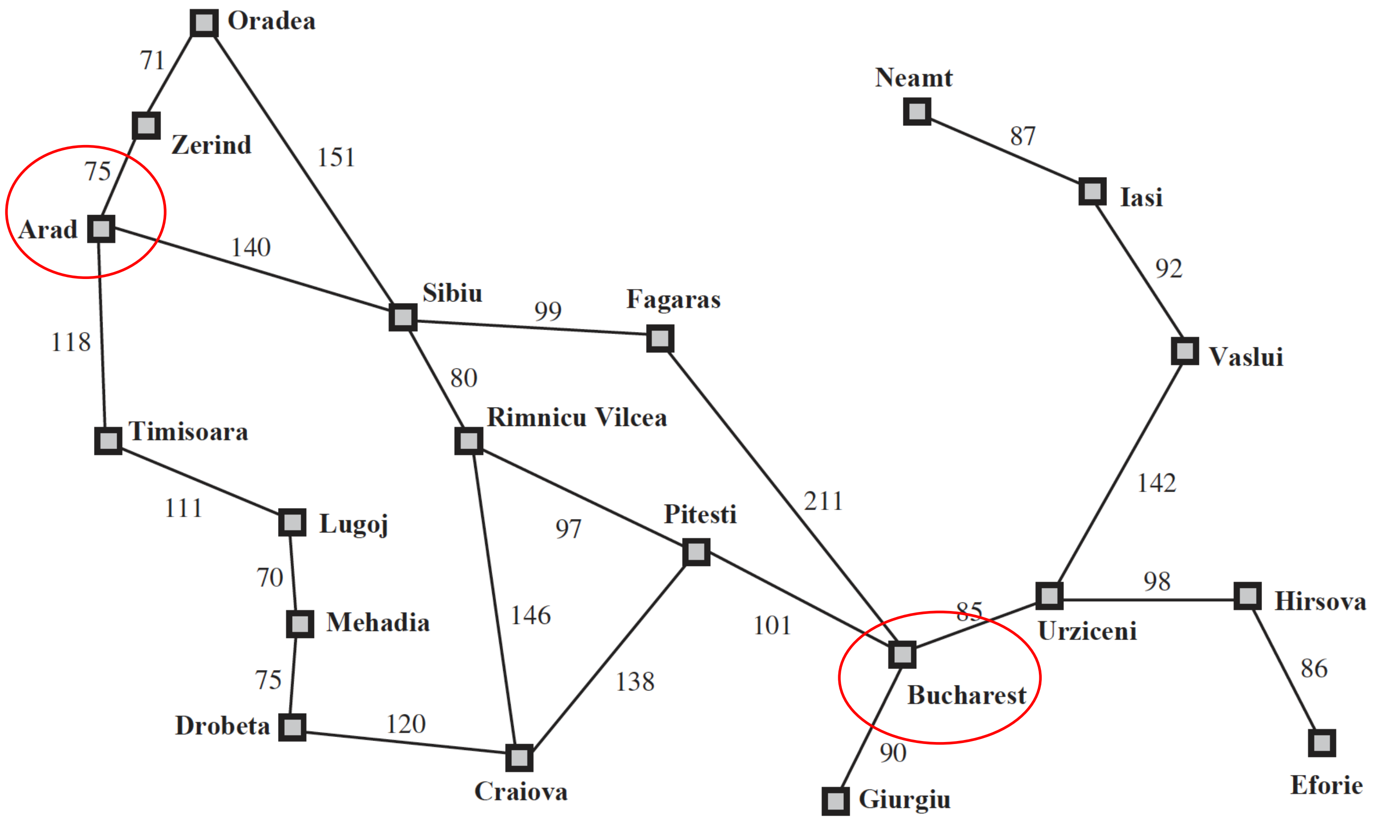
- Today:
  - Problem solving by search
    - ➢ Problem solving agents
    - ➢ Solutions and performance
    - ➢ Uninformed search strategies
    - ➢ Summary

Problem Solving Agents

- Problem solving agents:
  - find sequence of actions that achieve goals (e.g., maximize performance measure, minimize cost)

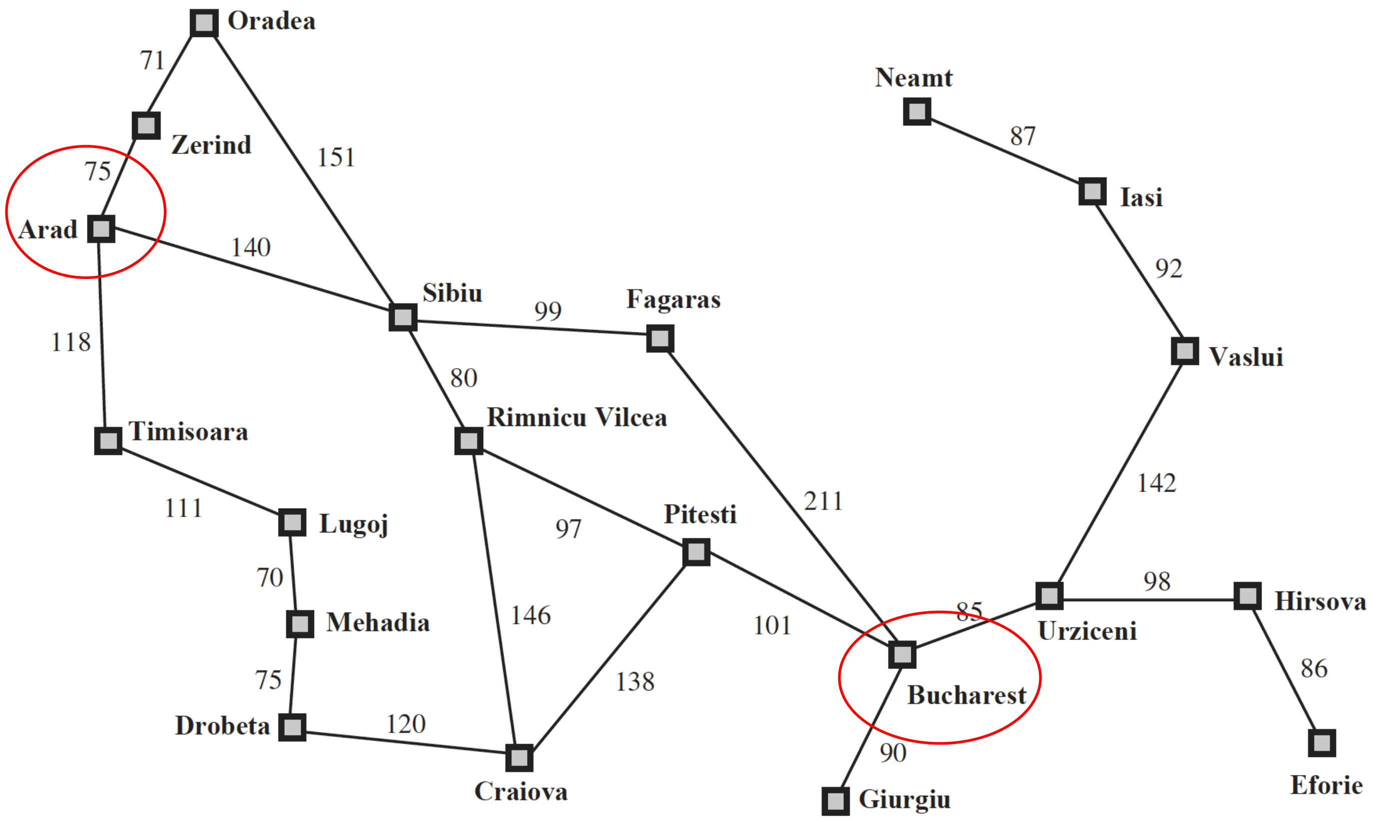
- Problem solving steps:
  1. Goal Formulation: where a goal is set of acceptable states
  2. Problem Formulation: choose the state space and action space
  3. Search
  4. Execute Found Solution

Problem Solving Agents



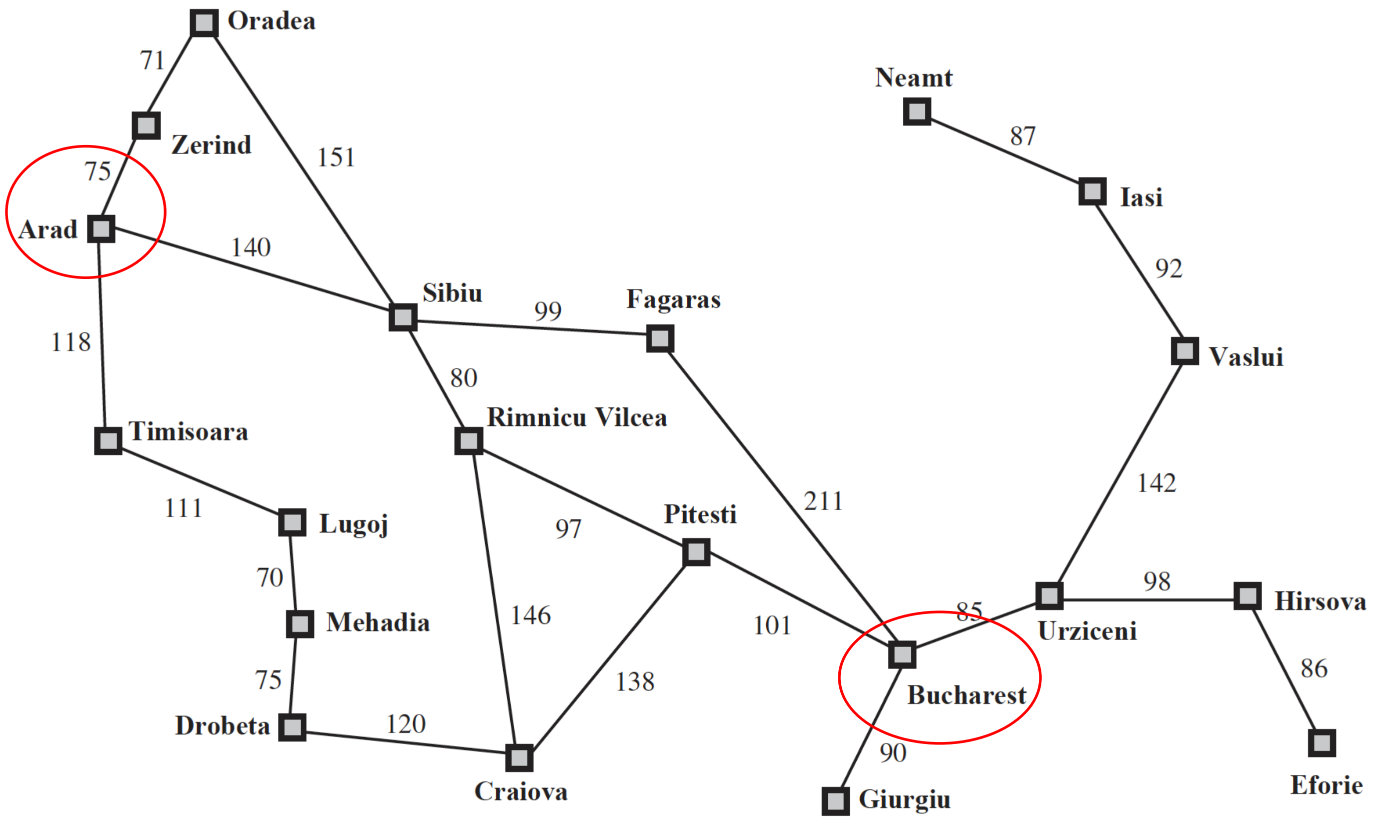
- An agent is in Arad and has a non-refundable ticket to fly out of Bucharest

Problem Solving Agents



- Goal formulation: reach Bucharest on time, e.g., goal state = *In(Bucharest)*
  - Goals help organize behavior by limiting the objectives that the agent is trying to achieve and hence the actions it needs to consider

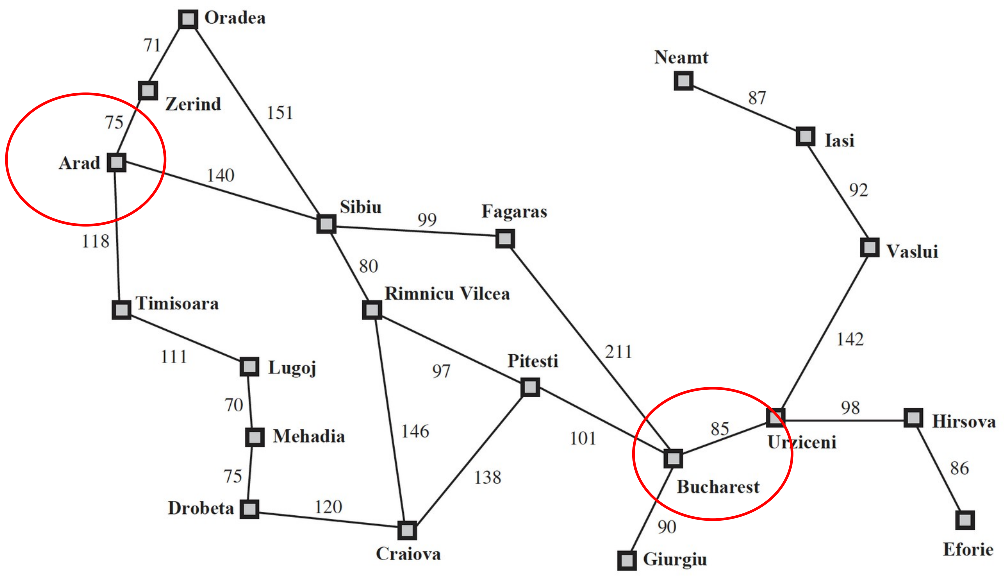
Problem Solving Agents



- Problem formulation: decide what actions and states it should consider
  - Action: driving from one town to one of its neighbor
  - State: being in a particular town

Problem Formulation

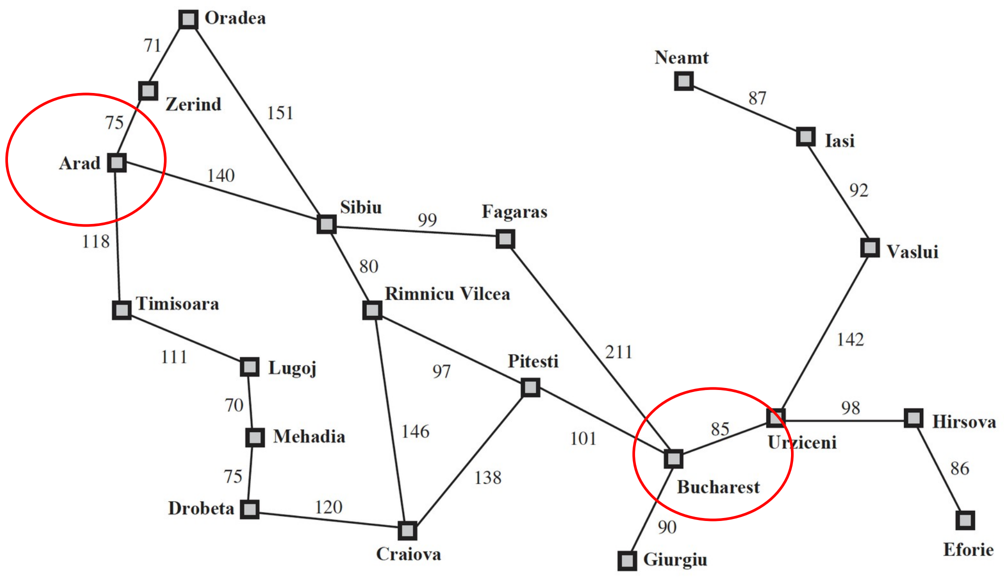
- Five components:
    1. Initial state and State space
    2. Action space
    3. Transition model
    4. Goal test
    5. Path cost



- Initial state: where the agent starts from
    - e.g., *In(Arad)*
- State space: the set of all reachable states from the initial state by any sequence of actions
    - e.g., {*In(Arad), In(Zerind), In(Sibiu),*… }

Problem Formulation

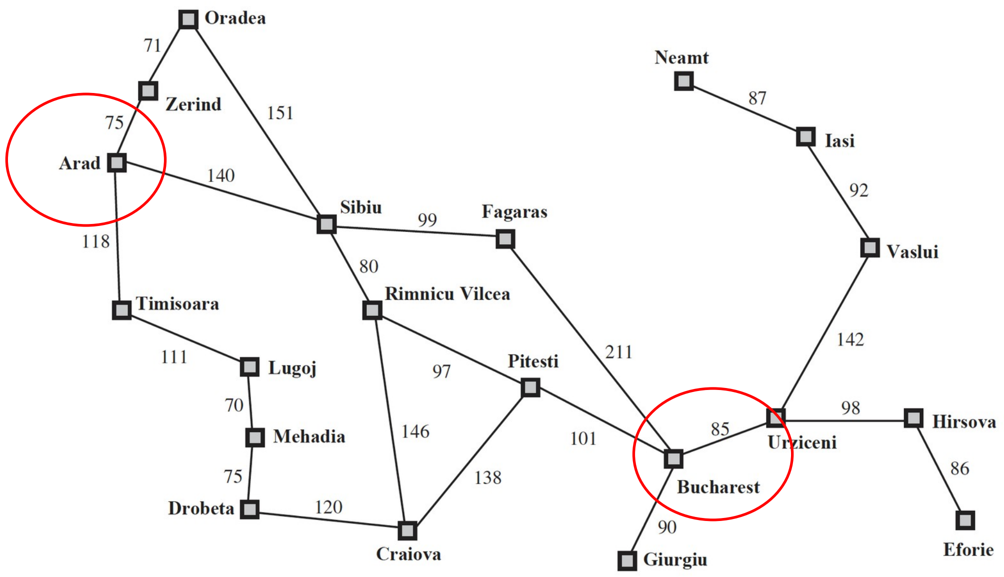
- Five components:
  1. State space and Initial state
  2. Action space
  3. Transition model
  4. Goal test
  5. Path cost



- Action space: the set of all possible actions the agent can take
  - e.g., action_space(state=*In(Arad)*)={*Go(Zerind), Go(Sibiu), Go(Timisoara)*}

Problem Formulation

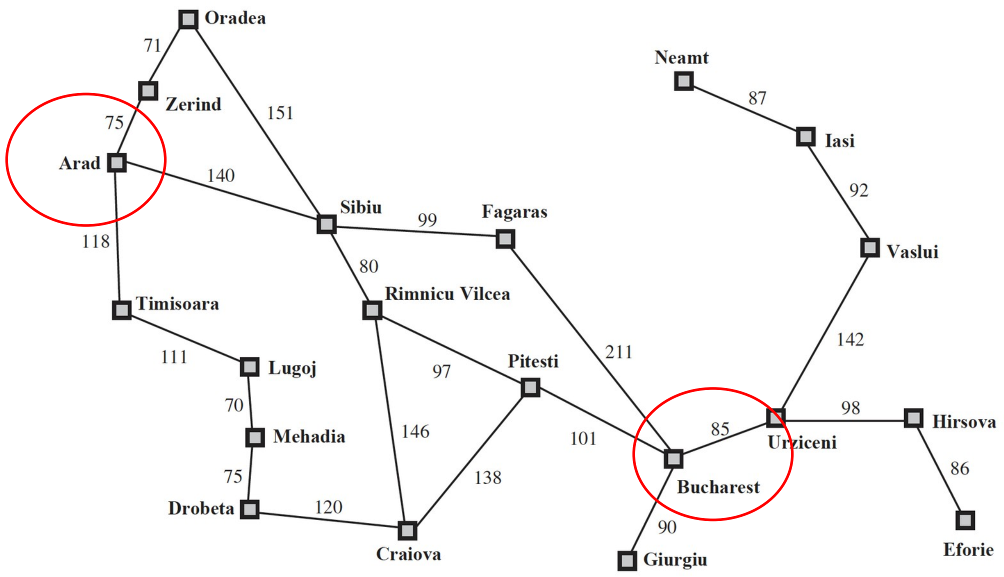
- Five components:
  1. State space and Initial state
  2. Action space
  3. Transition model
  4. Goal test
  5. Path cost



- Transition model: returns the next state *s'* that results from taking action *a* in current state *s*
  - e.g., RESULT(state=*In(Arad), Go(Zerind)*)=*In(Zerind)*

Problem Formulation

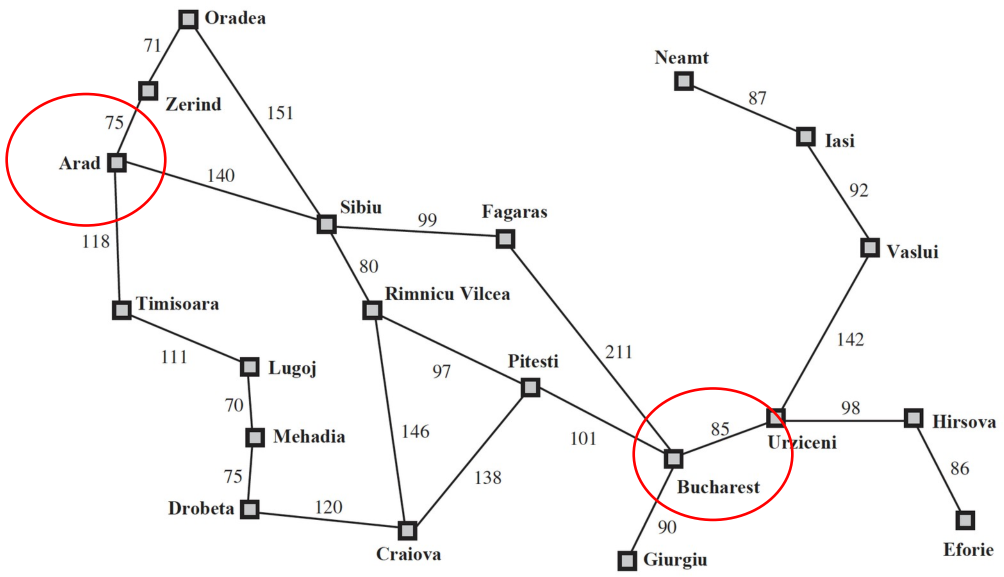
- Five components:
  1. State space and Initial state
  2. Action space
  3. Transition model
  4. Goal test
  5. Path cost



- Goal test: check if a given state is a goal state
  - e.g., check if current_state = *In(Bucharest)*

Problem Formulation

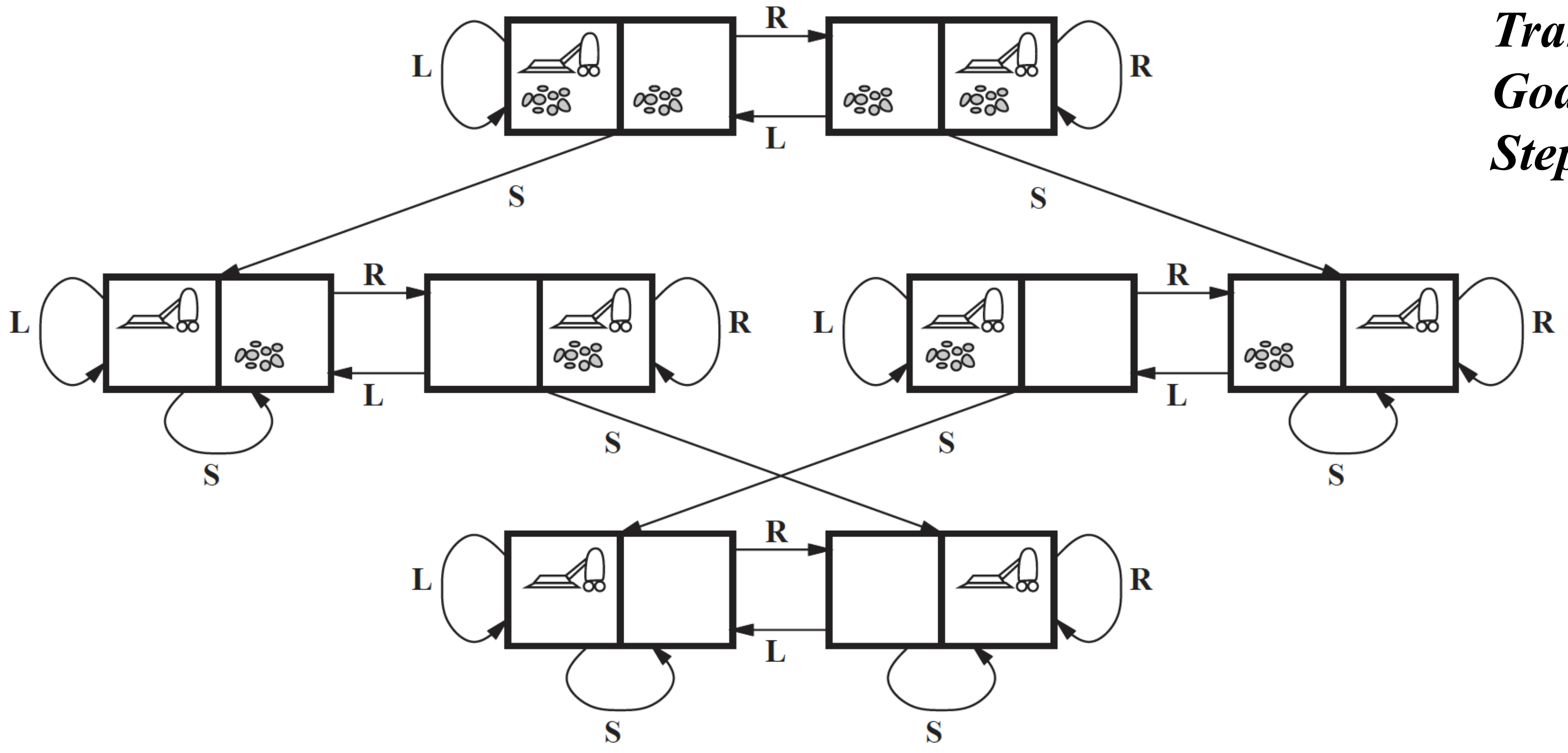
- Five components:
  1. State space and Initial state
  2. Action space
  3. Transition model
  4. Goal test
  5. Path cost



- Path cost: a numeric cost for a path (a sequence of states connected by a sequence of actions)
  - Step cost: cost(current state *s*, take action *a*, next state *s'*)
    - e.g., cost(*In(Arad), Go(Zerind), In(Zerind)*)=*75* (distance in km)
  - Path cost: sum of step costs along the path
    - E.g., path_cost({Arad, Sibiu, Fagaras})=*140+99=239*

Classical Problems

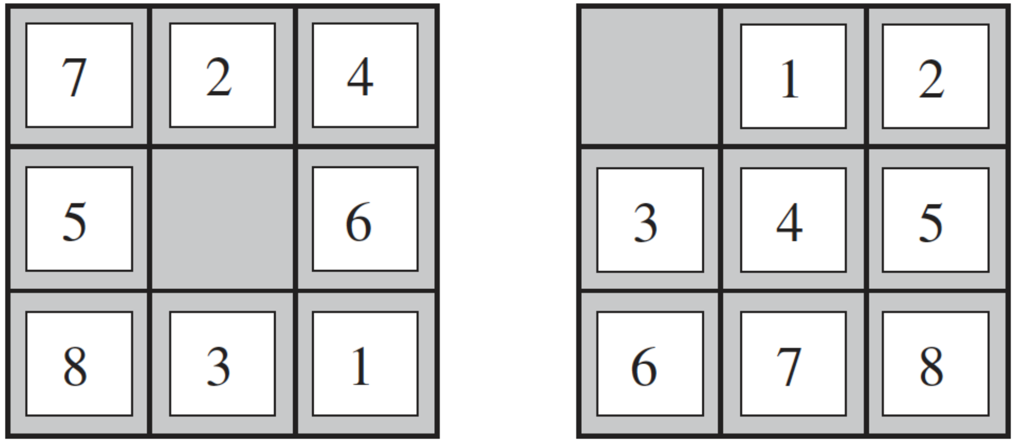
- Toy problem: Vacuum World



State: agent location and dirt location
Action: left, right, suck
Transition model: shown in the figure
Goal test: clean or not
Step cost: 1

Classical Problems

- Toy problem: 8-puzzle

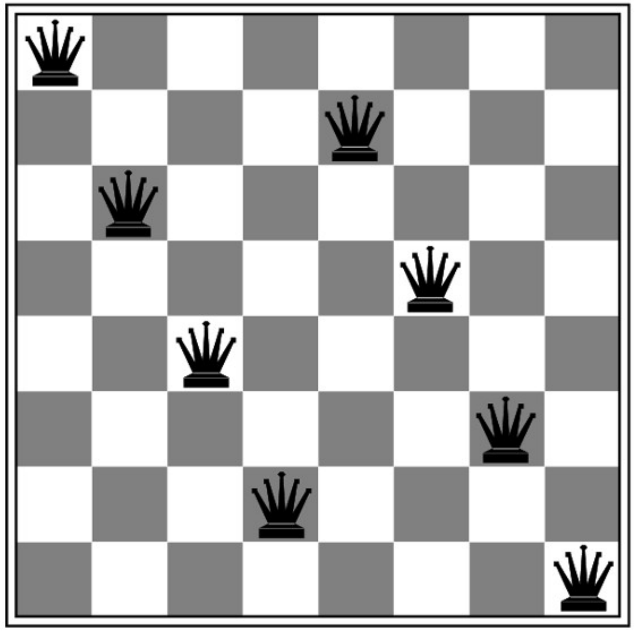


Start State Goal State

State: locations of tiles and blank space
Action: (move the space) left, right, up, down
Transition model: deterministic given current state and action
Goal test: match the goal configuration or not
Step cost: 1

Classical Problems

- Toy problem: 8-queens problem



State: any possible configurations of queens (), one per column in the leftmost columns, with no queen attacking another
Action: add a queen to any square in the leftmost empty column, such that it is not attacked by any other queens
Transition model: return the board with a queen added to the specified square
Goal test: 8 queens on the board, none attacked
Step cost: of no interest because only the final configuration matters

Classical Problems

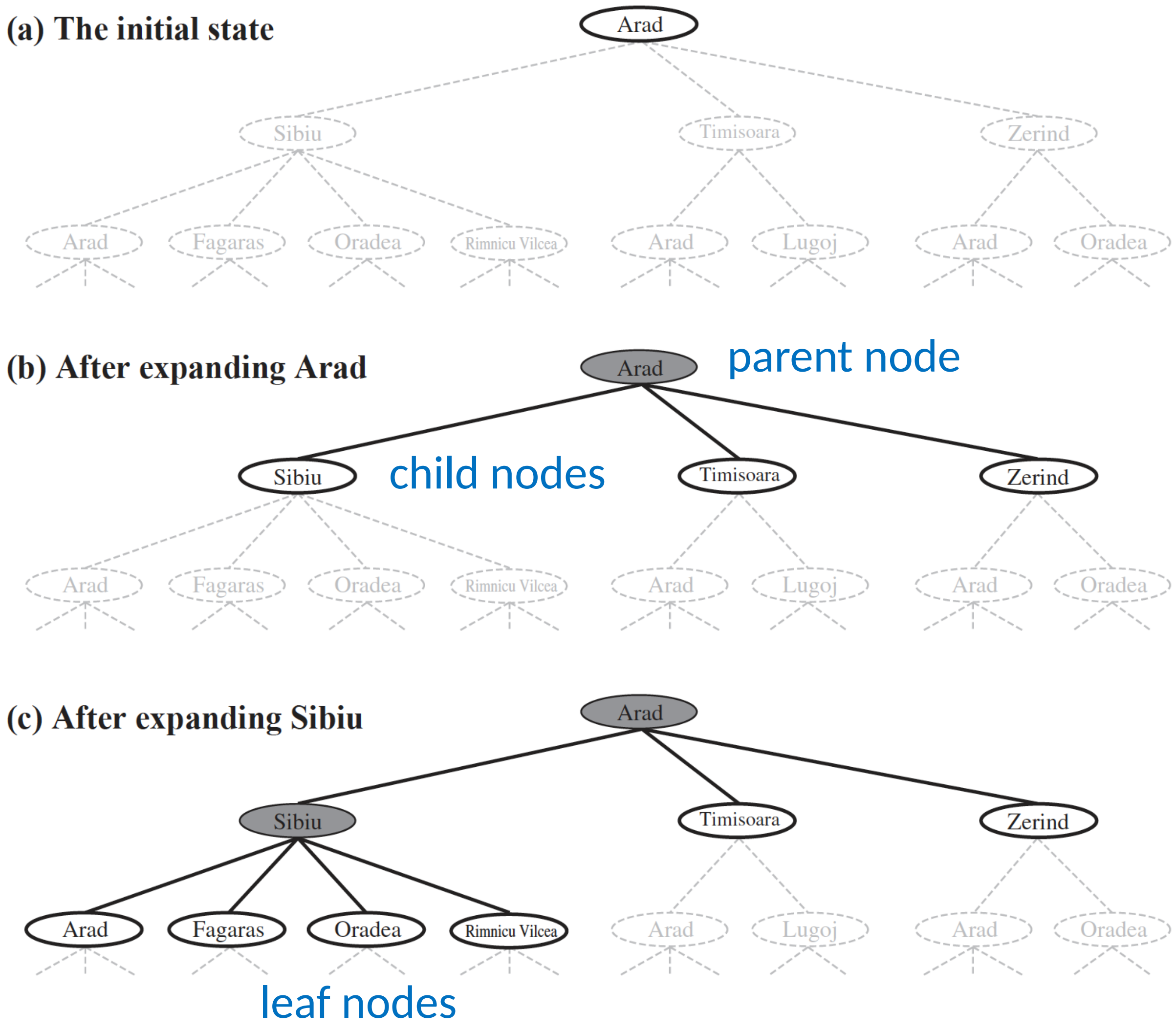
- Real-world problems
  - Route Finding Problem
  - Traveling Salesperson Problem (TSP)
  - Robot Navigation
  - Automatic Assembly Sequencing
  - Protein Design

Overview

- Problem solving by search
  - Problem solving agents
  - **Solutions and performance**
  - Uninformed search strategies
  - Avoiding repeated strategies
  - Partial information
  - Summary

Searching for Solutions

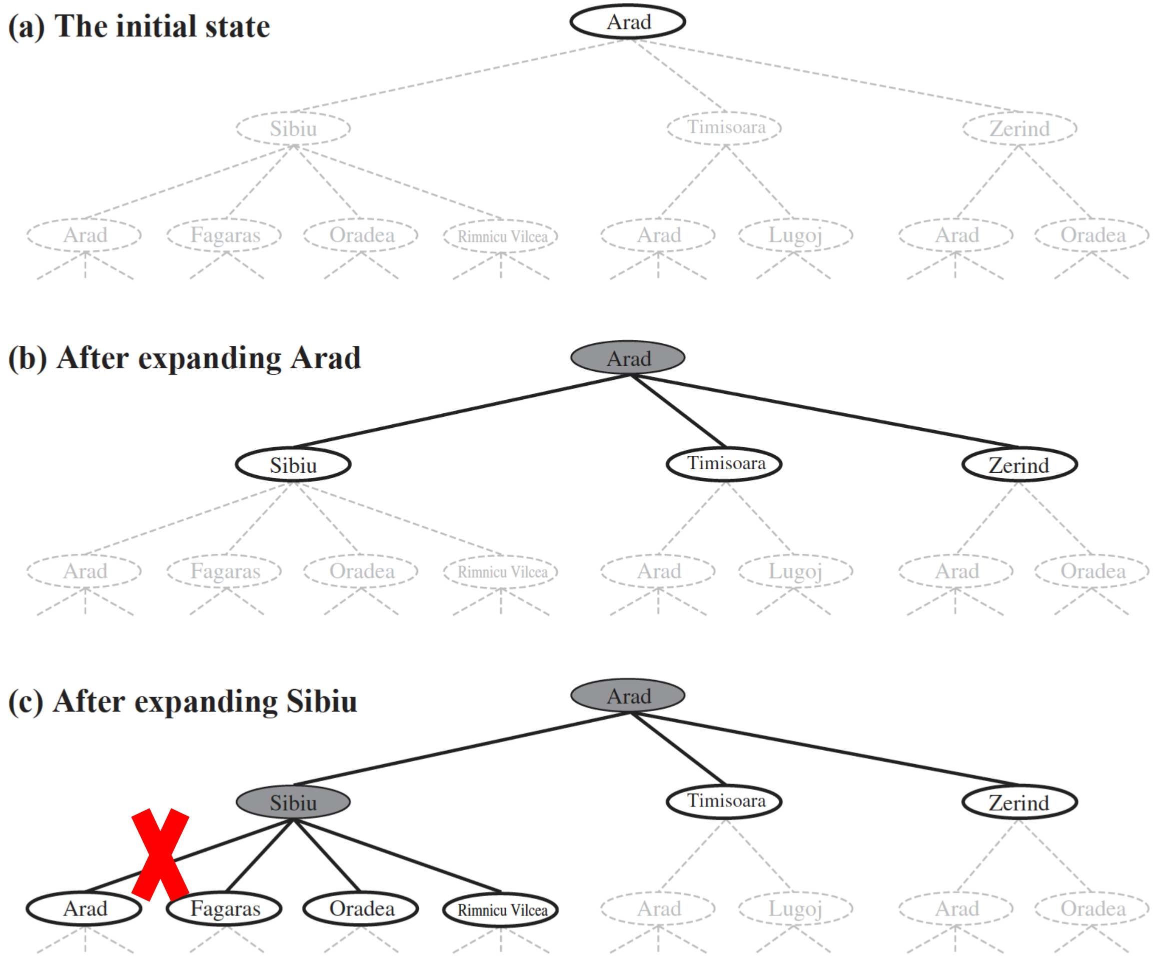
- Solution is a sequence of actions
- Search tree:
  - The possible action sequences starting from the initial state form a search tree with the initial state as the root
  - Branches correspond to actions, nodes correspond to states
  - Expand new nodes to grow the tree
- Search algorithms all have this basic structure; they vary in search strategy – choose which state to expand next

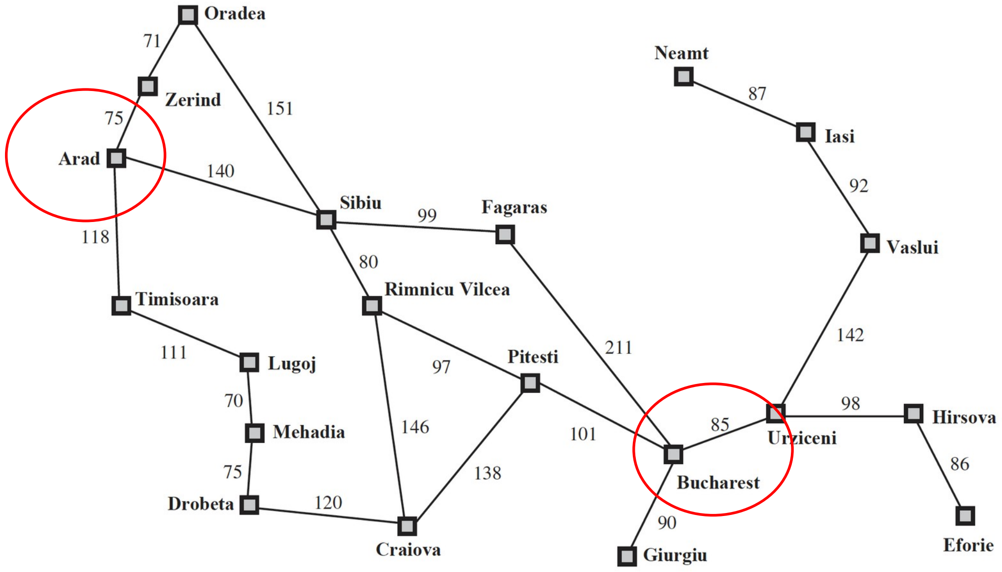


Frontier: set of lead nodes available for expansion

Avoid Looping & Repeated States

- Tree search vs Graph search
  - Graph search removes the redundant paths by introducing the explored set to remember the expanded nodes





Explored set={Arad, Sibiu}
Frontier={Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Zerind}

Common Data Structures in Searching

- Node of the tree:
  - :
  - :
  - :
  - (frequently):
  - (maybe):

- A queue to store the frontier
  - Pop(*queue*)
  - Insert(*element, queue*)
  - Order: FIFO, LIFO, Priority, …

Performance Measure

- Four elements of performance:
  - Completeness
  - Optimality
  - Time Complexity
  - Space Complexity

- Parameters for the complexity
  - Branching factor:
  - Depth of the shallowest goal node:
  - Maximum length of any path in the state space:

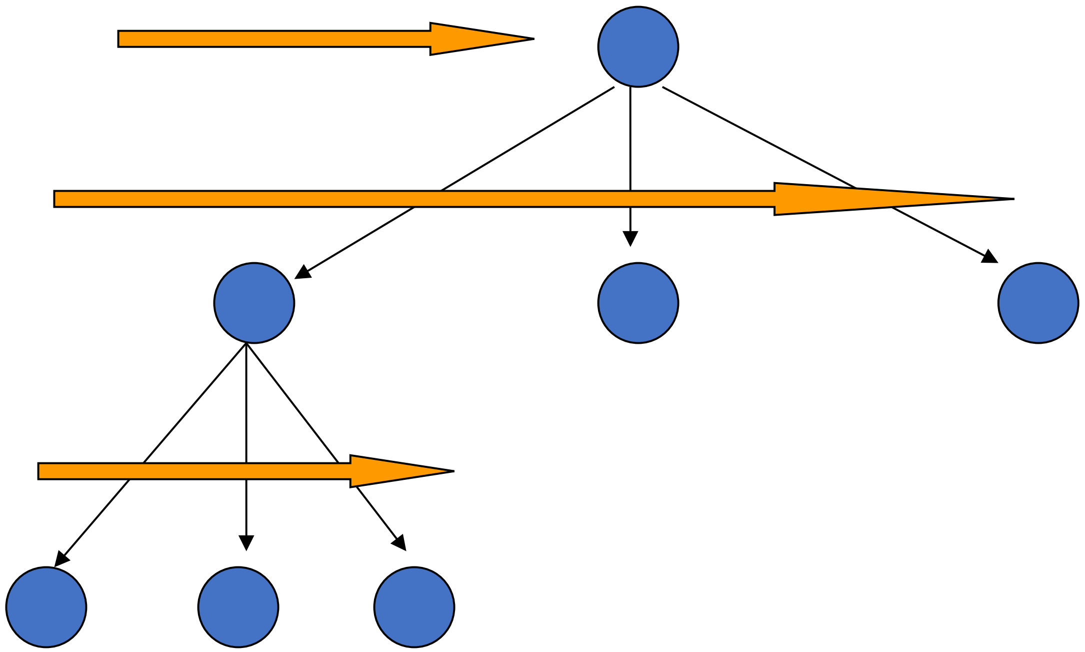
- Search cost and total cost

Overview

- Problem solving by search
  - ➢Problem solving agents
  - ➢Solutions and performance
  - ➢**Uninformed search strategies**
  - ➢Avoiding repeated strategies
  - ➢Partial information
  - ➢Summary

- Uninformed search
  - Generate successors and distinguish a goal state from a non-goal state
  - Search strategies differ in the order in which nodes are expanded

Breadth-first Search

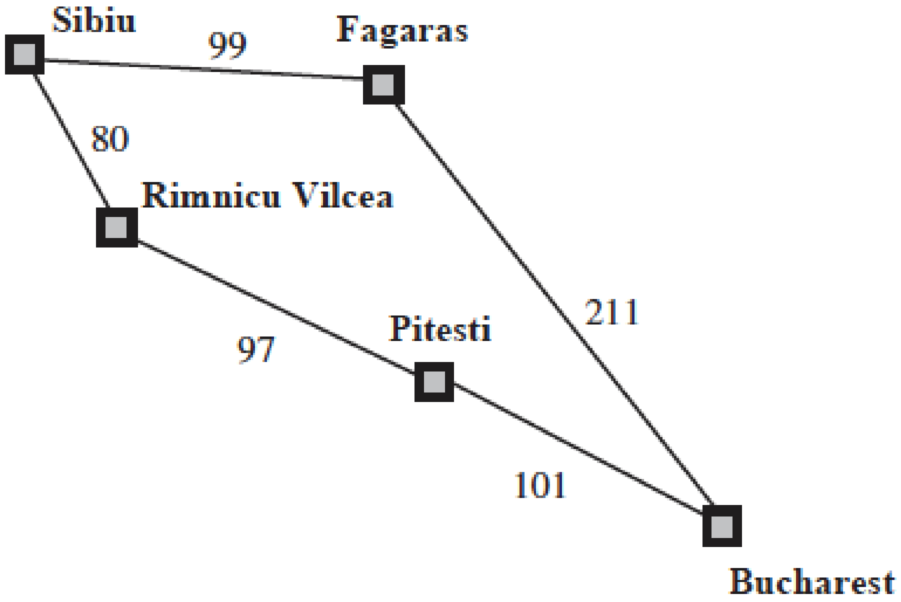


- Root is expanded first
- Then all successors at level 2
- Then all successors at level 3, etc.
- Goal test when a node is generated

- Properties:
  - Complete if and are finite
  - Optimal if path cost increases with depth
  - Time complexity and space complexity are both

Uniform-cost Search

- Path cost is important: always expand the node with lowest path cost
- Goal test when a node is selected for expansion

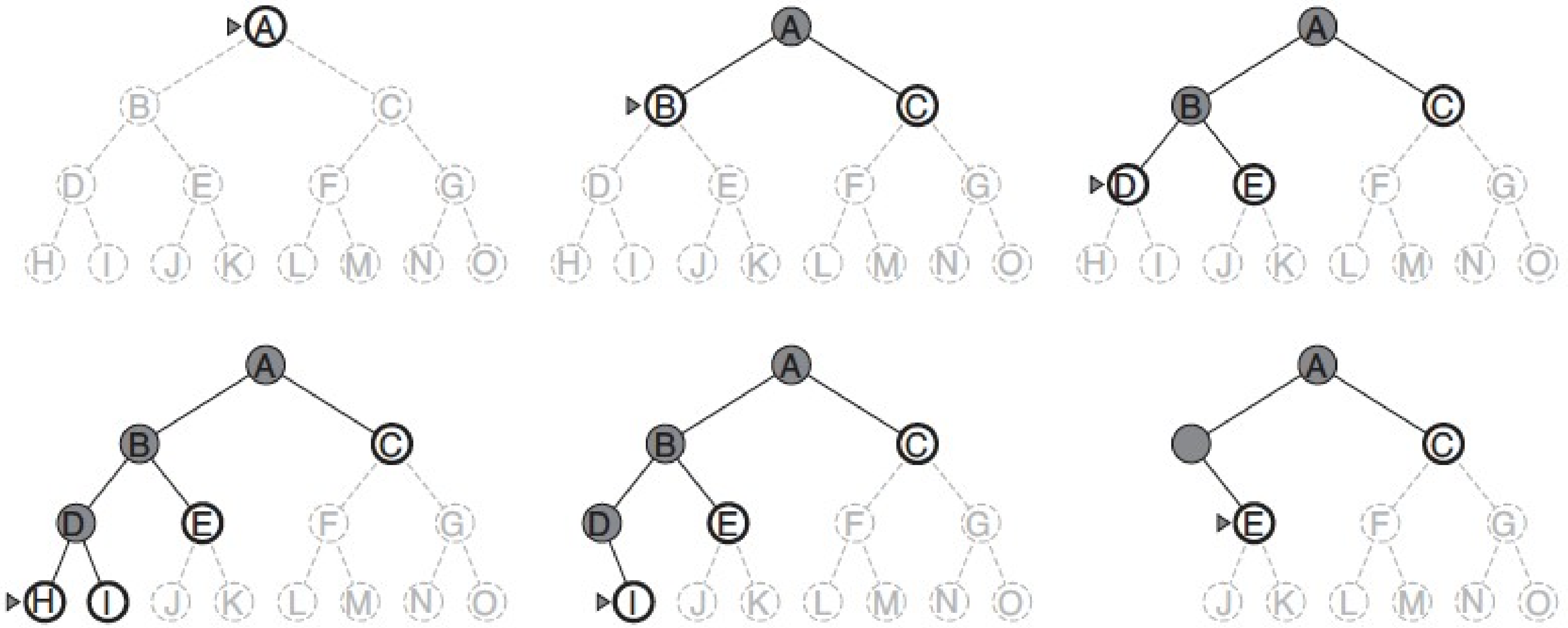


- Properties:
  - Complete if and are finite (positive step cost)
  - Optimal if path cost increases with depth
  - Cost:
    - Let denote the cost of the optimal solution
    - Assume that step cost is at least
    - Worst-case time and space complexity is
  - Could be worse than breadth-first search (similar cost when all step costs are equal)

Exponential complexity is always a big problem

Depth-first Search

- Always expand the deepest node at the bottom of the tree
- Search proceeds immediately to the deepest level
- Back up to the next deepest node that still has unexplored successors



Depth-first Search

- Properties:
  - Complete only for graph search in finite state spaces
  - Suboptimal
  - No clear advantage in terms of time complexity
  - Space complexity for tree-search version:
    - Only need to store a single path from the root to a leaf node and the remaining unexpanded sibling nodes for each node on the path
    - Backtracking even does better space-wise:

Depth-first search can fail embarrassingly in infinite state spaces

Depth-limited Search

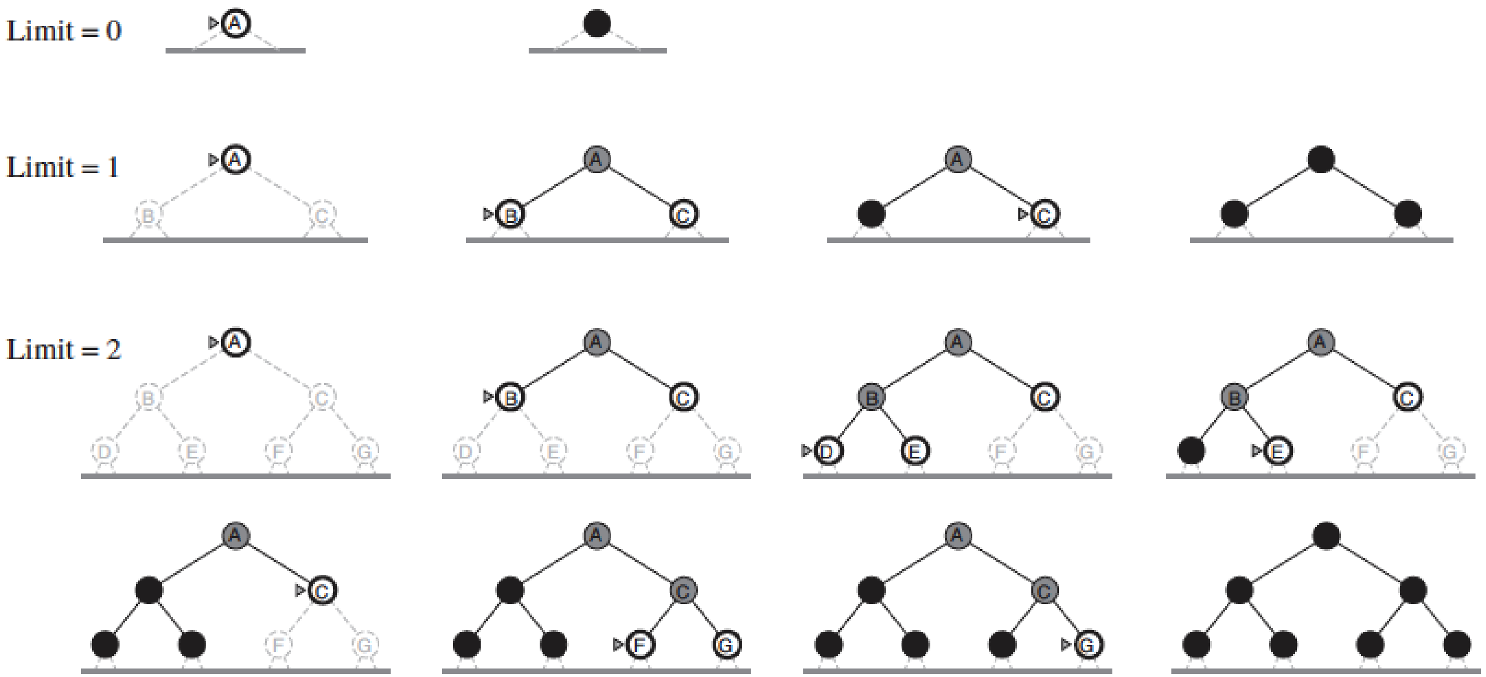
- Depth-first search can fail in infinite state spaces
- Like depth-first search but with depth limit
  - Solve the infinite-path problem

- Properties:
  - Incomplete if
  - Suboptimal if (similar to depth-first)
  - Time complexity is
  - Space complexity is

How to address the incompleteness and suboptimality?

Iterative Deepening Search

- A combination of depth and breadth-first search
- Gradually increase the limit until (a goal will be found)

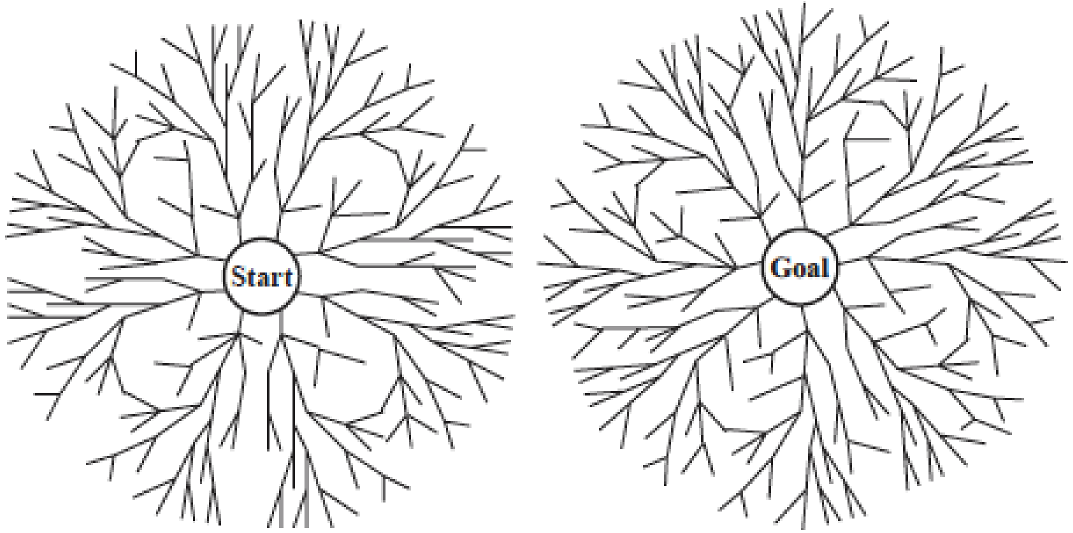


- Properties:
  - Complete if and are finite
  - Optimal if path cost increases with depth
  - Time complexity
  - Space complexity

In general, iterative deepening is the preferred uninformed search method when the search space is large and the depth of the solution is not known.

Bidirectional Search

- Previous methods still suffer from exponential time complexity
- Run two simultaneous searches
  - One forward from the initial state; the other backward from the goal
  - Check if the frontiers of the two searches intersect
- Properties (when both use breadth-first):
  - Complete if and are finite
  - Optimal if path cost increases with depth
  - Time complexity
  - Space complexity



Summary

- To search we need goal and problem formulation
- A problem has initial state, state and action spaces, transition model, goal test and path function
- Performance measures: completeness, optimality, time and space complexity
- Uninformed search has no additional domain specific knowledge:
  - Breadth-first
  - Uniform-cost
  - Depth-first
  - Depth-limited
  - Iterative deepening
  - Bidirectional