Problem Solving By Searching

- ***Problem Solving Agents***
- Solutions and Performance
- Uninformed Search Strategies
- Avoiding Repeated States/Looping
- Partial Information
- Summary

Problem Solving Agent

Problem-solving agents: find sequence of actions that achieve goals.

Problem-Solving Steps:

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

Preliminaries

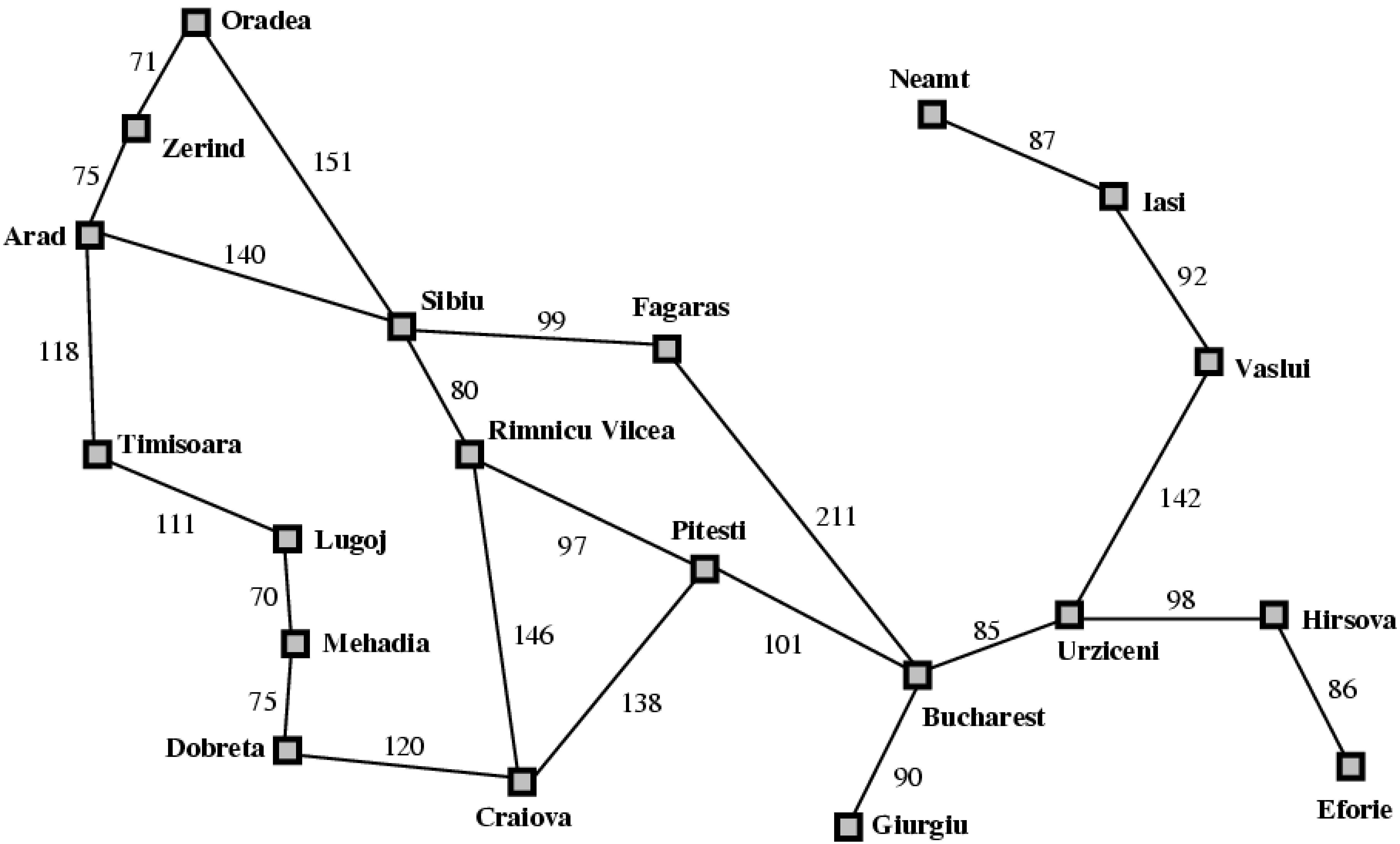
We need to define two things:

- Goal Formulation
  - ✓ Define objectives
- Problem Formulation
  - ✓ Define actions and states

Problem Formulation State Space Search

Four components:

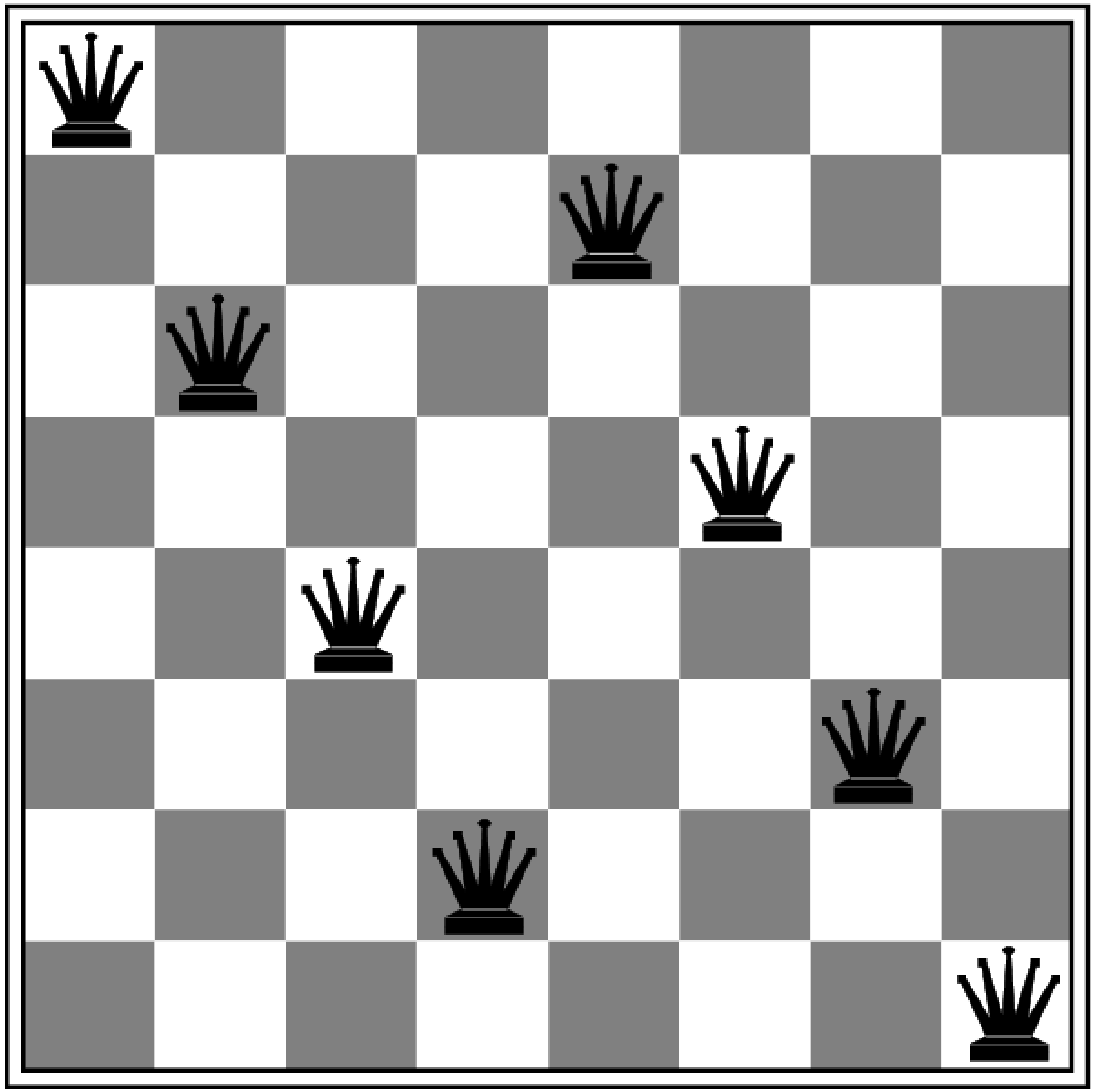
1. The initial state
2. Actions (successor function)
3. Goal test
4. Path Cost


Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Dobreta
120
Craiova
Neamt
87
Iasi
92
Vaslui
142
Urziceni
98
Hirsova
85
Bucharest
86
90
Giurgiu
Eforie

Other Examples

Toy Problems:

- ✓ Vacuum World
- ✓ 8-puzzle
- ✓ 8-queens problem



Other Examples

Real Problems

- ✓ Route-Finding Problem
- ✓ Robot Navigation
- ✓ Automatic Assembly Sequencing
- ✓ Protein Design
- ✓ Internet Searching

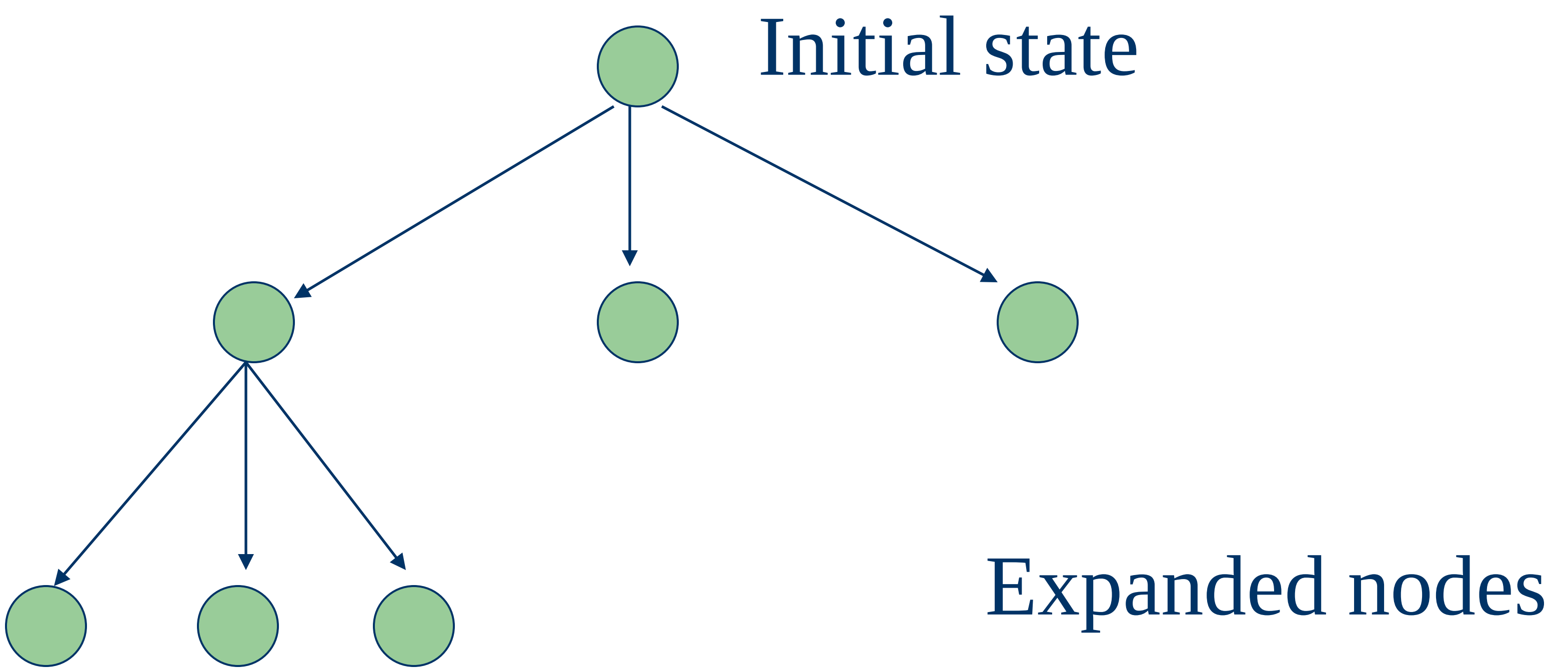
Problem Solving By Searching

- Introduction
- ***Solutions and Performance***
- Uninformed Search Strategies
- Avoiding Repeated States
- Partial Information
- Summary

Solutions

- We search through a search tree
- We expand new nodes to grow the tree
- There are different search strategies
- Nodes contain the following:
  - ✓ state
  - ✓ parent node
  - ✓ action
  - ✓ path cost
  - ✓ *maybe* depth

Search Tree



Performance

Four elements of performance:

- Completeness (guaranteed to find solution)
- Optimality (optimal solution?)
- Time Complexity
- Space Complexity

Performance

Complexity requires three elements:

a. Branching factor **b**
b. Depth of the shallowest goal node **d**
c. Maximum length of any path **m** in the state space

Problem Solving By Searching

- Introduction
- Solutions and Performance
- ***Uninformed Search Strategies***
- Avoiding Repeated States
- Partial Information
- Summary

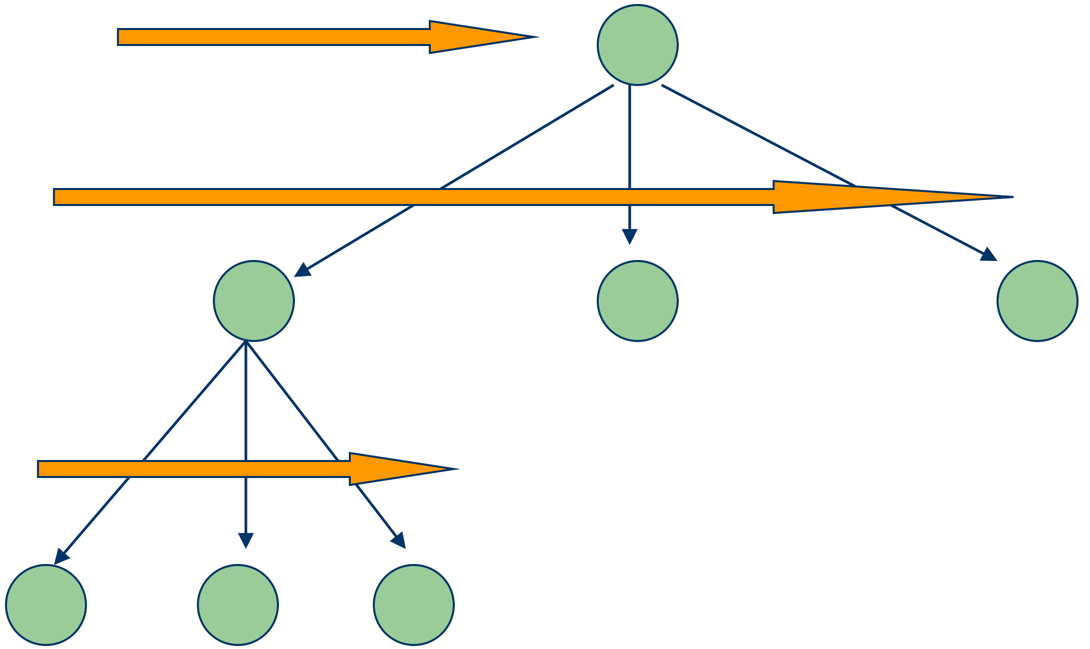
Breadth-First Search

- Root is expanded first
- Then all successors at level 2.
- Then all successors at level 3, etc.

Properties:

- Complete (if b and d are finite)
- Optimal (if path cost increases with depth)
- Cost is $O(b^{d+1})$
- Storage Let $n=b^{d+1}$: $O(n)$

Search



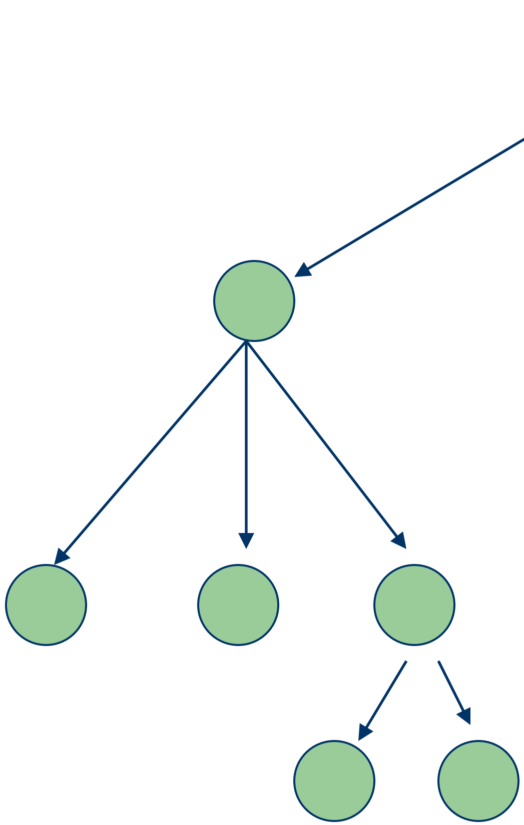
Uniform-Cost Search

- Expansion Approach---expands nodes completely
- Strategy: Expand node with lowest path cost.

Properties:

- Complete (if b and d are finite)
- Optimal (if path cost increases with depth)
- Cost: Similar to Breadth-first Search
- Could be worse than breadth first search

Search



Depth-First Search

- Expand the deepest node at the bottom of the tree.

Backtracking even does better space-wise: **O(d)**

Properties:

Only store the nodes on the current path including their unexpanded sibling nodes

- Incomplete
- suboptimal
- Space complexity is only O(**bd**)

Tree Depth First Search with Depth Bound L

Space Complexity Backtracking: O(d)

Space Complexity Expansion Depth-first Search: O(b*d)

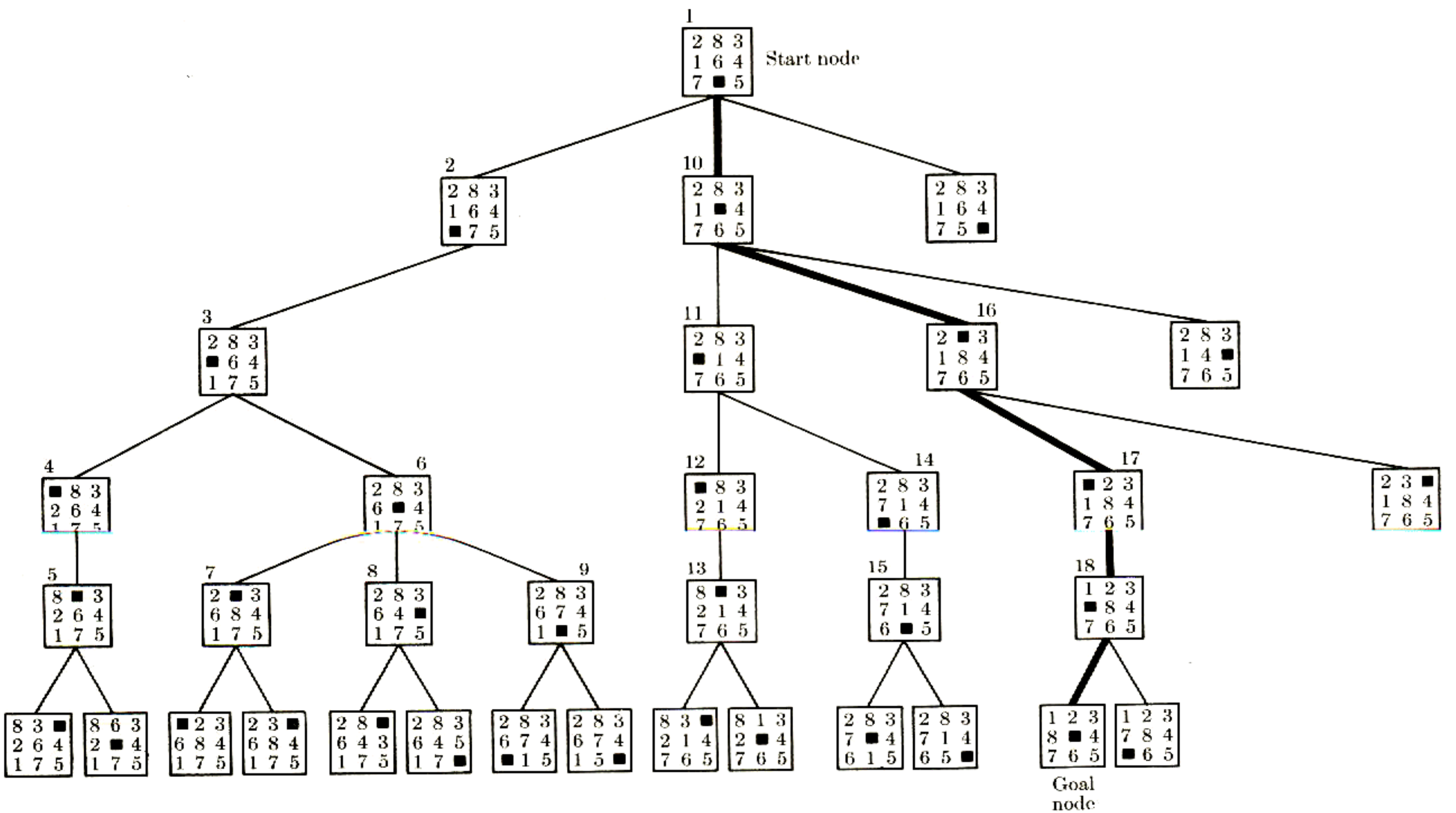
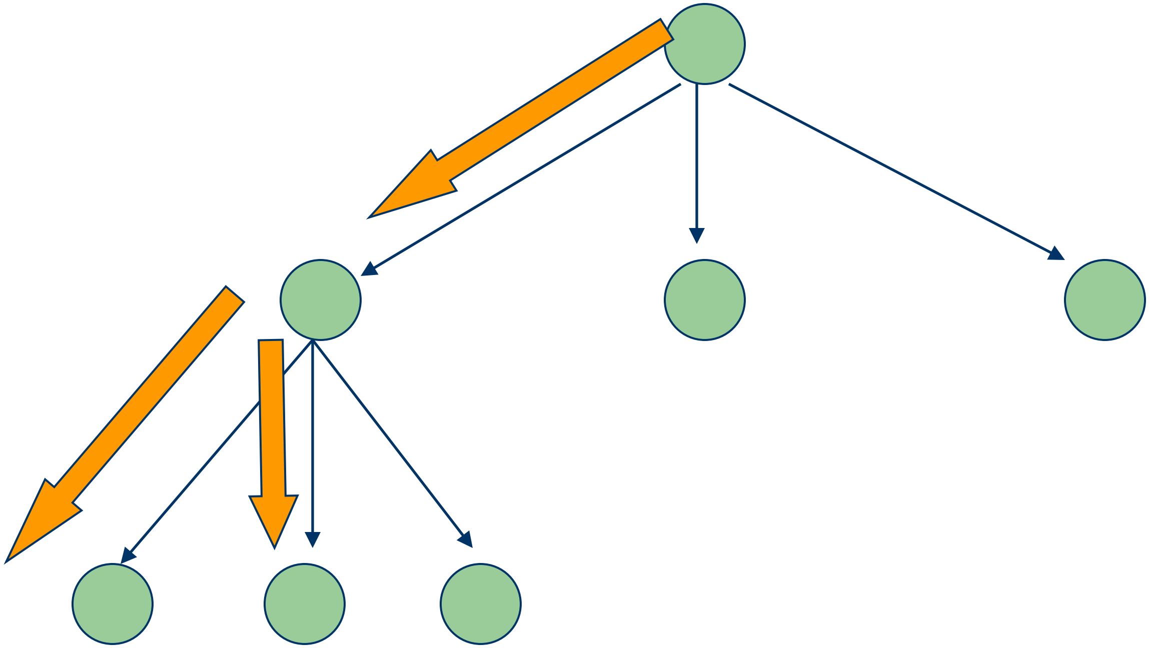


FIG. 3-5 *The tree produced by a depth-first search.*

Search



Depth-Limited

- Like depth-first search but with depth limit **L**.

Properties:

- Incomplete (if $\mathbf{L} < \mathbf{d}$)
- nonoptimal (if $\mathbf{L} > \mathbf{d}$)
- Time complexity is $O(\mathbf{b}^{L})$
- Space complexity is $O(\mathbf{bL})$

Iterative Deepening

- A combination of depth and breadth-first search.
- Gradually increases the limit **L**

Properties:

- Complete (if b and d are finite)
- Optimal if path cost increases with depth
- Time complexity is O($\mathbf{b}^{\mathbf{d}}$)

Problem Solving By Searching

- Introduction
- Solutions and Performance
- Uninformed Search Strategies
- ***Avoiding Repeated States/Looping***
- Partial Information
- Summary

Avoiding Looping & Repeated States (relates to expansion search)

- Use a list of expanded states; non-expanded states (*open* and *close list*)
- Use domain specific knowledge
- Use sophisticated data structures to find already visited states more quickly.
- Checking for repeated states can be quite expensive and slow down the search alg.

Remark: Non-expansion search strategies have to keep track of what operators have already been applied to avoid looping.

Problem Solving By Searching

- Introduction
- Solutions and Performance
- Uninformed Search Strategies
- Avoiding Repeated States
- ***Partial Information***
- Summary

Partial Information

Knowledge of states or actions is incomplete.

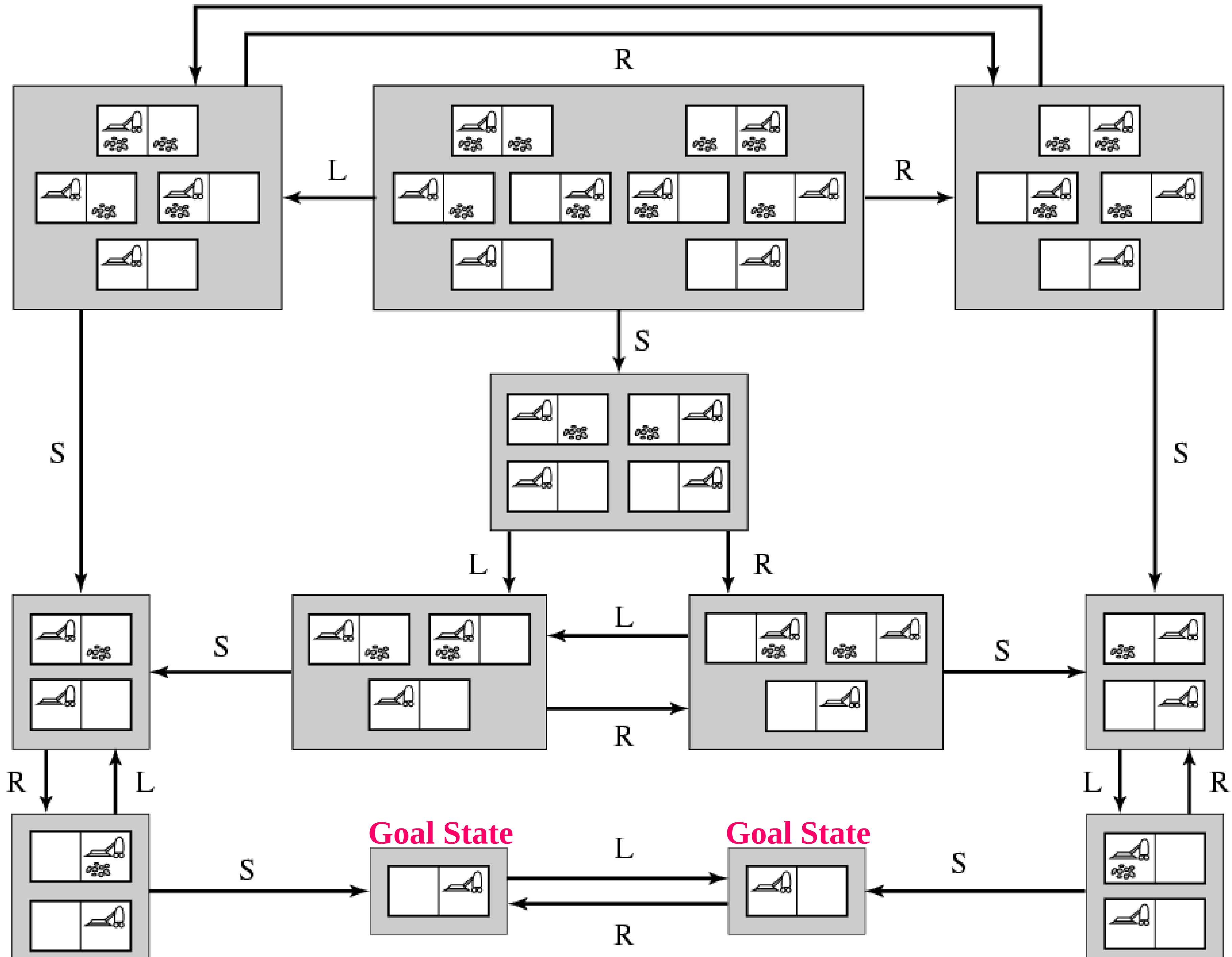
a. Sensorless problems
b. Contingency Problems
c. Exploration Problems

Problem types

Deterministic, accessible $\Longrightarrow$ *single-state problem*
Deterministic, inaccessible $\Longrightarrow$ *multiple-state problem*

Nondeterministic, inaccessible $\Longrightarrow$ *contingency problem*
must use sensors during execution
solution is a *tree* or *policy*
often *interleave* search, execution

Unknown state space $\Longrightarrow$ *exploration problem* ("online")



R
L
R
S
S
S
L
R
L
S
S
R
R
L
L
R
Goal State
Goal State
L
S
S
R

Problem Solving By Searching

- Introduction
- Solutions and Performance
- Uninformed Search Strategies
- Avoiding Repeated States
- Partial Information
- ***Summary***

Summary

- To search we need goal and problem formulation.
- A problem has initial state, actions, goal test, and path function.
- Performance measures: completeness, optimality, time and space complexity.

Summary

- Uninformed search has no additional domain specific knowledge: breadth and depth-first search, depth-limited, iterative deepening, bidirectional search.
- In partially observable environments, one must deal with uncertainty and incomplete knowledge.