Artificial Intelligence

Lecture 03: Solving Problems by Searching

Dr. Bilal Jan

Department of Computer Science / NUCES-FAST, Peshawar

Spring 2026

Outline

Problem-Solving Agents

Artificial Intelligence | Lecture 03

Representation of Agent's states

Atomic:

- State is a black box with no internal structure
- Simple but uninformative

Factored:

- State is a vector of attribute values (Boolean, real-valued, or symbolic)
- Allows analysis of individual components

Structured:

- State includes objects with attributes and relationships to other objects
- Most expressive representation

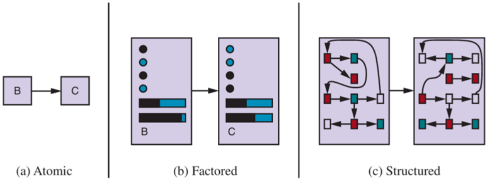
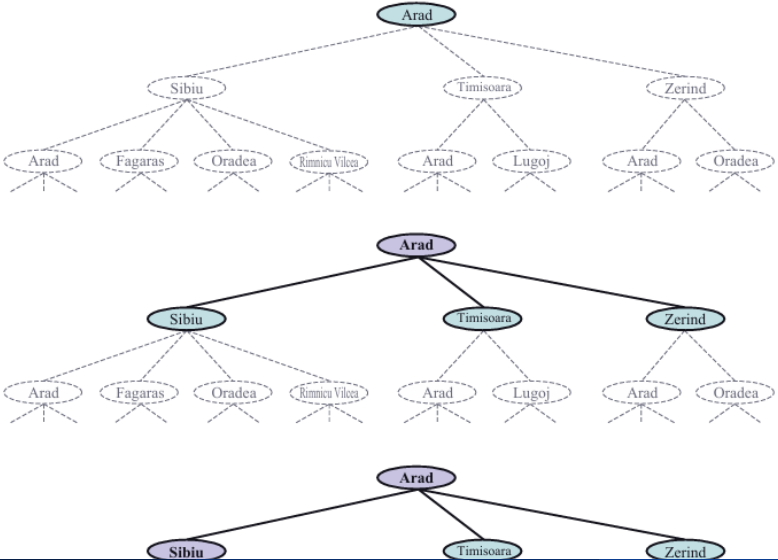


Figure: Atomic, Factored and Structured

Agent's Environment: Goal from Arad → Bucharest



Problem-Solving Agents

- **Assumed environment**: Single agent, episodic, fully observable, deterministic, static, discrete, and known.
- **Problem-solving agent**: Plans ahead to find action sequences leading to goals
- Uses **atomic representations**: States treated as wholes, no internal structure
- **Four-phase process**:
  1. **Goal formulation**: Adopt objectives to achieve
  2. **Problem formulation**: Describe states and actions
  3. **Search**: Simulate sequences to find solution
  4. **Execution**: Execute actions in solution

Example: Route Finding in Romania

Scenario:

- Agent in Arad
- Goal: Reach Bucharest
- Has map of Romania
- Must plan route

Solution:

- Search through possible routes
- Find: Arad $\rightarrow$ Sibiu $\rightarrow$ Fagaras $\rightarrow$ Bucharest
- Execute actions

Open-loop vs. Closed-loop

Open-loop: Execute without monitoring percepts (deterministic)
Closed-loop: Monitor percepts during execution (nondeterministic)

Search Problems: Formal Definition

A search problem consists of:

1. **State space**: Set of possible states
2. **Initial state**: Starting state
3. **Actions**: Available actions in each state
4. **Transition model**: Result of applying action to state
5. **Goal test**: Check if state is a goal
6. **Action cost function**: Cost of each action

- **Solution**: Sequence of actions from initial to goal state
- **Optimal solution**: Solution with lowest path cost

Search Problems: Formal Definition

- The **actions** available to the agent. Given a state s, $\textsc{Actions}(s)$ returns a finite[2] set of actions that can be executed in s. We say that each of these actions is **applicable** in s. An example: Action Applicable

$$\textsc{Actions}(Arad) = \{ToSibiu, ToTimisoara, ToZerind\}.$$

- A **transition model**, which describes what each action does. $\textsc{Result}(s,a)$ returns the state that results from doing action a in state s. For example, Transition model

$$\textsc{Result}(Arad, ToZerind) = Zerind.$$

- An **action cost function**, denoted by $\textsc{Action-Cost}(s,a,s')$ when we are programming or $c(s,a,s')$ when we are doing math, that gives the numeric cost of applying action a in state s to reach state s'. A problem-solving agent should use a cost function that reflects its own performance measure; for example, for route-finding agents, the cost of an action might be the length in miles (as seen in Figure 3.1), or it might be the time it takes to complete the action. Action cost function

Figure: Ref. section 3.1.1 from book

The Vacuum Cleaner: State space

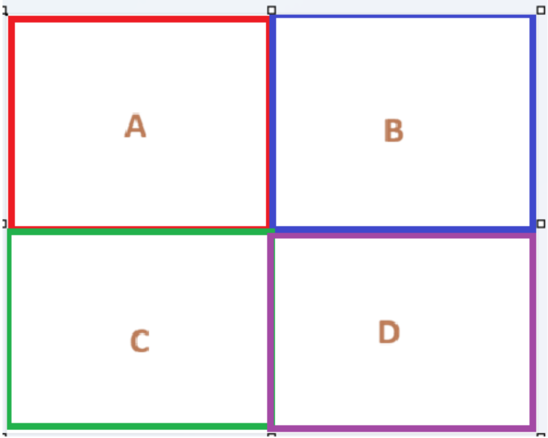
```
Agent at LEFT      Agent at RIGHT
---------------------------------
(L, D, D)  --R-->  (R, D, D)
   | S                     | S
(L, C, D)  --R-->  (R, C, D)


(L, D, C)  --R-->  (R, D, C)
   | S                     | S
(L, C, C)  --R-->  (R, C, C)
```

Figure: State space

Extended Vacuum world: 4 squares

What you think about the state space? Assume 4 squares.



4 square vacuum world.

- Locations: $Loc = \{A, B, C, D\}$
- Dirt status per square: $\{CLEAN, DIRTY\}$
- Any state is 5 tuple: $s = (loc, a, b, c, d)$
- Here: $loc \in Loc$ and $a, b, c, d \in \{CLEAN, DIRTY\}$
- $S = Loc \times \{CLEAN, DIRTY\}^4$
- State space: $|S| = 4 \times 2^4 = 64$
- n-square world?

Sliding Puzzles

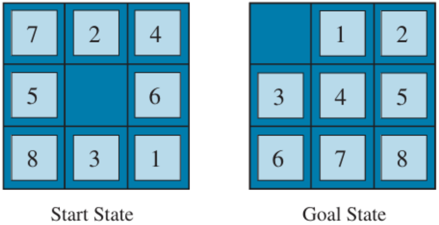


Figure: Sokoban 3x3: $9! = 362,880$

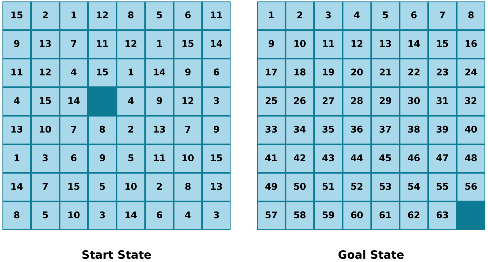
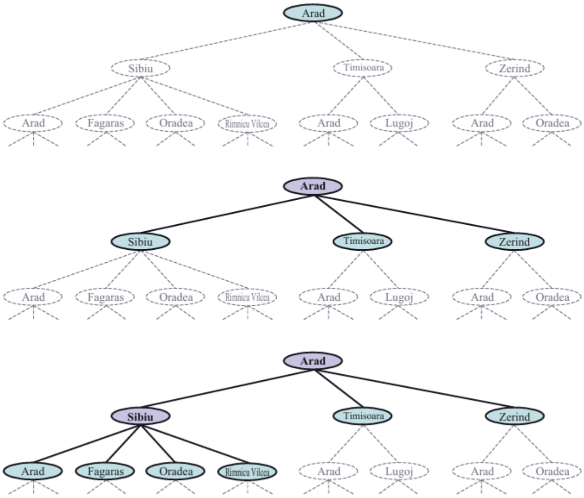


Figure: 8x8 Puzzle

State Space–Graph–Search–Tree



SEARCH TREE
(Each node has ONE unique path to root)

STATE SPACE
(Multiple paths possible)

Search Algorithms

Artificial Intelligence | Lecture 03

Best-First Search Framework

- **General approach**: Choose node with minimum evaluation function $f(n)$
- **Key data structures**:
  - **Node**: Contains STATE, PARENT, ACTION, PATH-COST
  - **Frontier**: Priority queue of nodes to expand
  - **Reached**: Set/table of visited states
- **Algorithm**:
  1. Choose node with minimum $f(n)$ from frontier
  2. If goal, return solution
  3. Otherwise, expand node and add children to frontier

Redundant Paths

- **Problem**: Multiple paths can reach same state
- **Cycle**: Path that returns to previous state
- **Redundant path**: Worse way to reach same state
- **Three approaches**:
  1. **Graph search**: Remember all reached states
  2. **Tree-like search**: Don't check for redundant paths
  3. **Cycle checking**: Check only for cycles

Measuring Search Performance

Four evaluation criteria:

1. **Completeness**: Guaranteed to find solution?
2. **Cost optimality**: Finds lowest-cost solution?
3. **Time complexity**: How long to find solution?
4. **Space complexity**: How much memory needed?

Complexity measures:

- b: Branching factor; d: Depth of shallowest solution; m: Max depth

Uninformed Search

Artificial Intelligence | Lecture 03

Uninformed Search Strategies

- **Uninformed search**: No clue about distance to goal
- Only use problem definition (states, actions, costs)
- **Main algorithms**: BFS, UCS, DFS, DLS, IDS, Bidirectional

Breadth-First Search (BFS)

- **Strategy**: Expand shallowest nodes first (FIFO)
- **Properties**:
  - Complete: Yes; Optimal: Yes (unit costs)
  - Time: $O(b^d)$; Space: $O(b^d)$
- **Disadvantages**: Exponential space complexity

Uniform-Cost Search (UCS)

- **Strategy**: Expand node with lowest path cost $g(n)$
- **Properties**:
  - Complete: Yes; Optimal: Yes
  - Time/Space: $O(b^{1+\lfloor C^*/\epsilon \rfloor})$
- Uses **late goal test**: Check when node expanded

Depth-First Search (DFS)

- **Strategy**: Expand deepest unexpanded node first (LIFO)
- **Properties**:
  - Complete: No; Optimal: No
  - Time: $O(b^m)$; Space: $O(bm)$ (Linear!)

Depth-Limited & Iterative Deepening Search

Depth-Limited Search:

- DFS with depth limit l
- Avoids infinite paths
- Space: $O(bl)$

Iterative Deepening:

- Repeatedly apply DFS with increasing limits
- Complete and optimal (unit costs)
- Space: $O(bd)$

Bidirectional Search

- **Idea**: Search forward from start and backward from goal
- **Motivation**: $b^{d/2} + b^{d/2} \ll b^d$
- **Properties**: Time $O(b^{d/2})$, Space $O(b^{d/2})$

Comparison of Uninformed Search

Algorithm	Complete?	Optimal?	Time	Space
BFS	Yes	Yes	$O(b^d)$	$O(b^d)$
UCS	Yes	Yes	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$
DFS	No	No	$O(b^m)$	$O(bm)$
IDS	Yes	Yes	$O(b^d)$	$O(bd)$

Informed Search

Artificial Intelligence | Lecture 03

Informed (Heuristic) Search

- **Informed search**: Uses domain-specific hints (heuristics)
- **Heuristic function** $h(n)$: Estimated cost from n to goal
- Example: Straight-line distance in route-finding

Greedy Best-First Search

- **Evaluation function**: $f(n) = h(n)$
- Expands node that appears closest to goal
- **Properties**: Not optimal, can get stuck in loops

A* Search

- **Evaluation function**: $f(n) = g(n) + h(n)$
- **Properties** (with admissible h):
  - Complete: Yes; Optimal: Yes
  - Optimally efficient: Expands fewest nodes
  - Space: $O(b^d)$ (Main limitation)

Admissible and Consistent Heuristics

Admissible Heuristic

$h(n)$ never overestimates the cost to reach goal: $h(n) \leq h^*(n)$

Consistent (Monotonic) Heuristic

For every node n and successor n': $h(n) \leq c(n, a, n') + h(n')$

Weighted A* Search

- **Evaluation function**: $f(n) = g(n) + W \times h(n)$ where $W > 1$
- **Satisficing search**: Accept "good enough" solutions faster
- Finds solution with cost between C^* and $W \times C^*$

Memory-Bounded Search

Artificial Intelligence | Lecture 03

Memory-Bounded Search Algorithms

- **Problem**: A* uses too much memory
- **Main algorithms**: Beam search, IDA*, RBFS, SMA*
- **Tradeoff**: Save memory by revisiting states

Beam Search & IDA*

- **Beam Search**: Keep only k best nodes in frontier (Incomplete)
- **IDA***: Iterative deepening with f-cost cutoff
  - Complete and optimal; Space: $O(bd)$

Heuristic Functions

Artificial Intelligence | Lecture 03

Designing Heuristic Functions

- **Relaxed problems**: Fewer restrictions (e.g., Manhattan distance)
- **Pattern databases**: Precompute costs for subproblems
- **Landmarks**: Precompute distances to major intermediate states

Summary

Artificial Intelligence | Lecture 03

Chapter 3 Summary

- **Uninformed search**: BFS, UCS, DFS, IDS (Blind search)
- **Informed search**: A*, Greedy, Weighted A* (Uses $h(n)$)
- **Heuristics**: Crucial for performance; must be admissible for optimality
- **Memory**: Main bottleneck for A*; solved by IDA*/RBFS