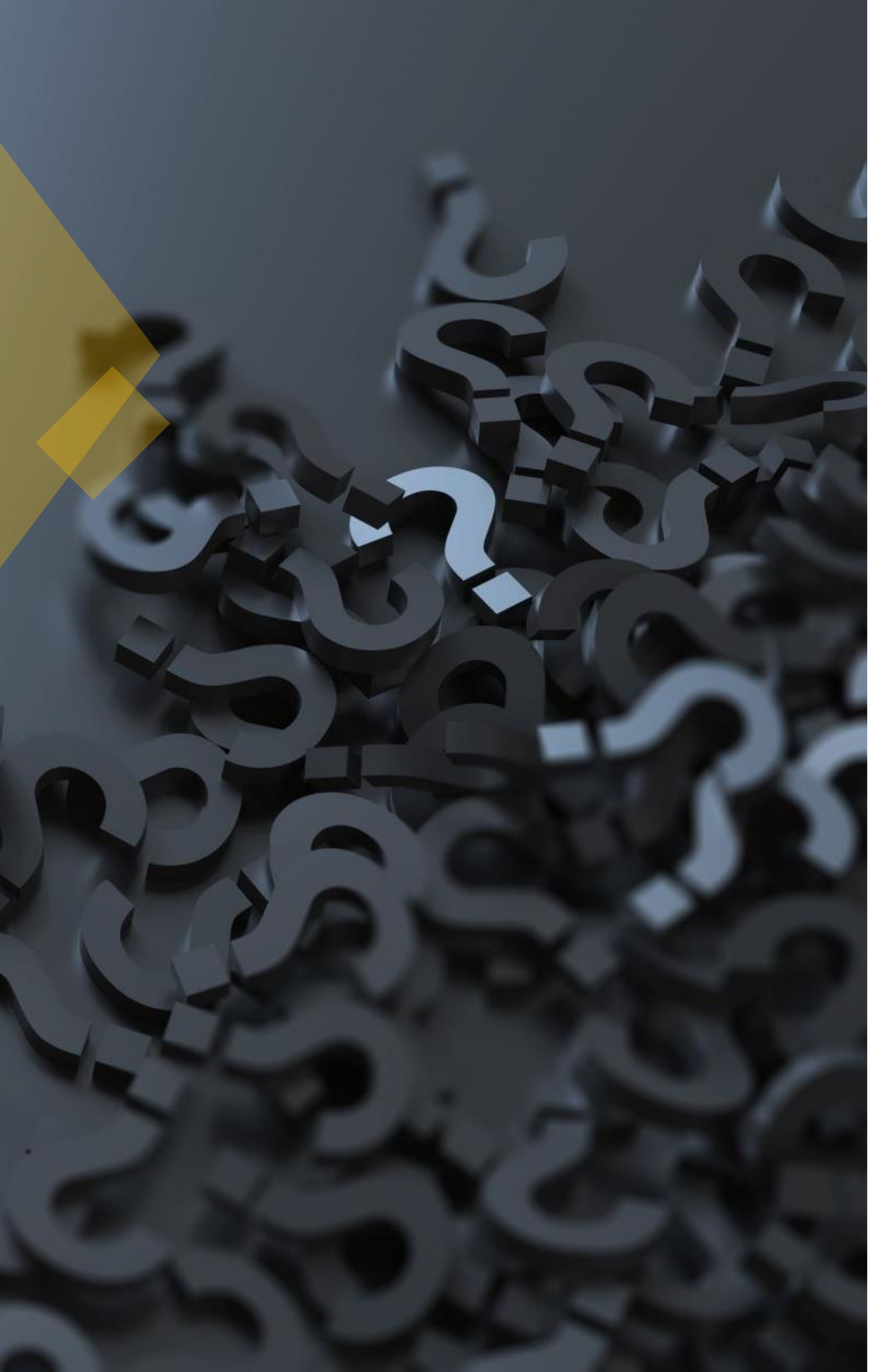
Solving Problems by Searching

Ahmed Ibrahim



Agenda

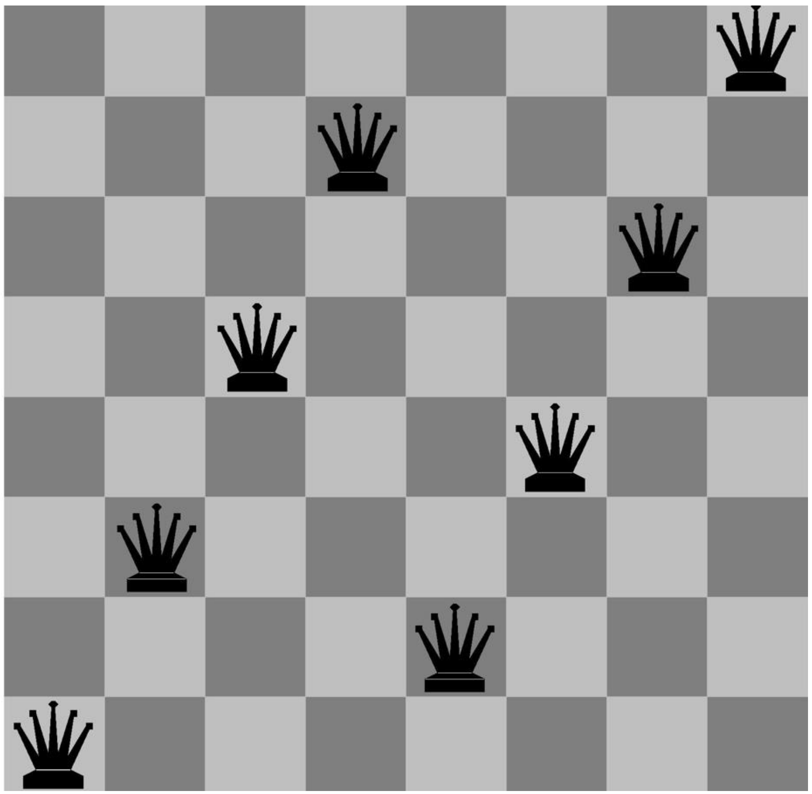
- Problem solving agents
- Problem formulation
- Real-world Examples
- State space vs. Search space
- Search strategies

Goal-based agents

- Reflex agents: use a mapping from states to actions.
- Goal-based agents: problem solving agents or planning agents.

Goal-based agents

- Agents that work towards a goal.
- Agents consider the impact of actions on future states.
- Agent's job is to identify the action or series of actions that lead to the goal.
- Formalized as a search through possible solutions.



The 8-queens

- The 8-queen problem: on a chess board, place 8 queens so that no queen is attacking any other horizontally, vertically or diagonally.
- Number of possible sequences to investigate:

  $64 * 63 * 62 * \ldots * 57 = 1.8 \times 10^{14}$

Problem solving as search

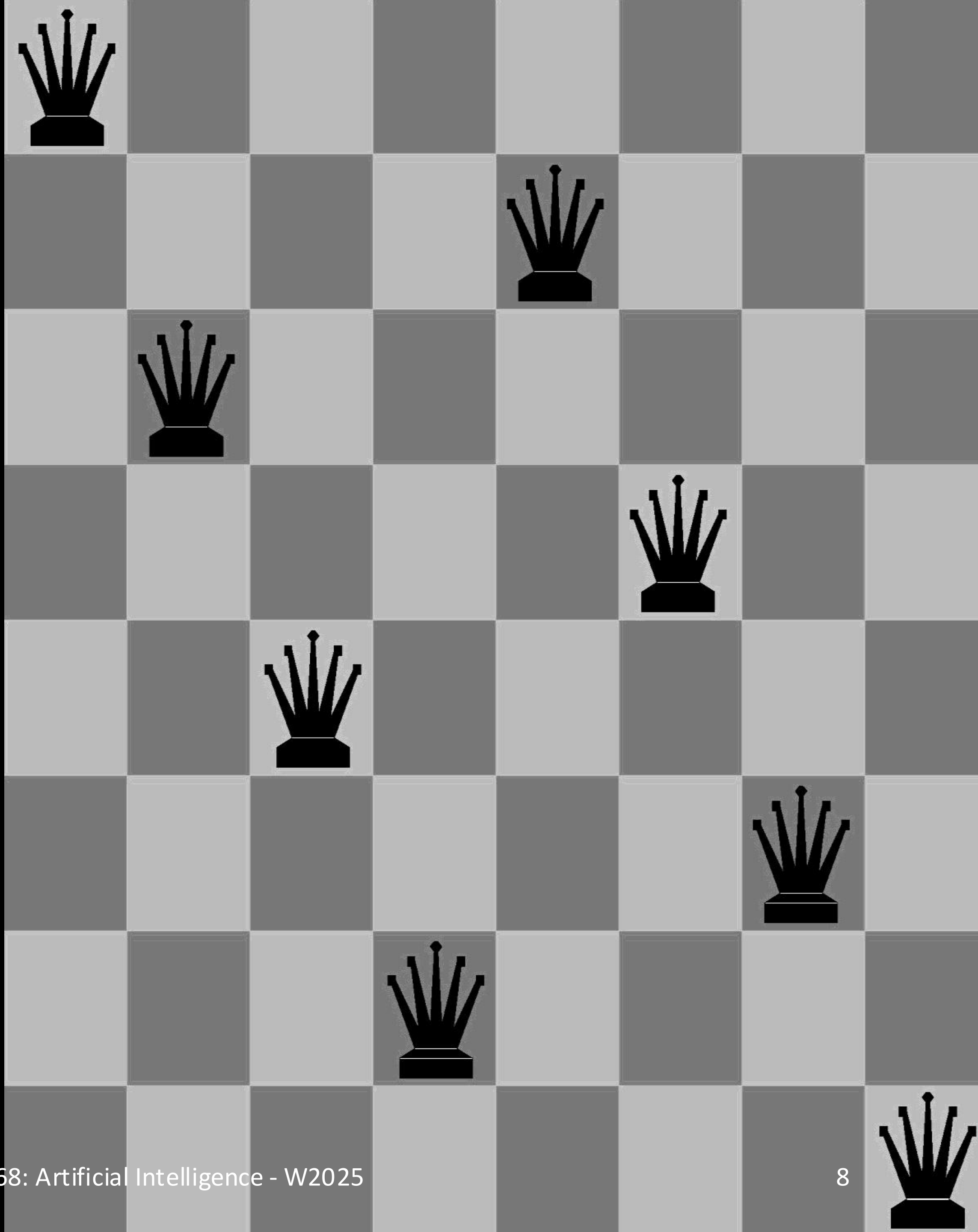
- Define the problem through:
  - (a) Goal formulation
  - (b) Problem formulation
- Solving the problem as a 2-stage process:
  - (a) Search: “mental” or “offline” exploration of several possibilities
  - (b) Execute the solution found

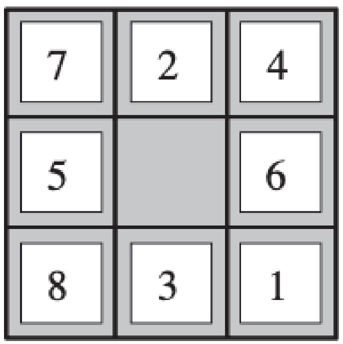
Problem formulation

- **Initial state**: the state in which the agent starts
- **States**: All states reachable from the initial state by any sequence of actions (State space)
- **Actions**: possible actions available to the agent. At a state *s*, *Actions*(*s*) returns the set of actions that can be executed in state s. (Action space)
- **Transition model**: A description of what each action does *Results*(*s, a*)
- **Goal test**: determines if a given state is a goal state
- **Path cost**: function that assigns a numeric cost to a path w.r.t. performance measure

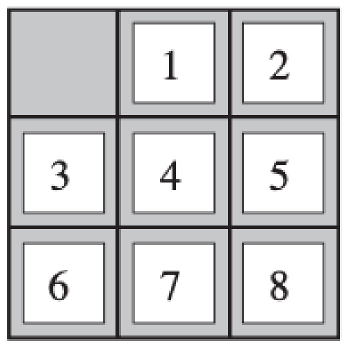
The 8-queen Problem

- **States** - all arrangements of Zero to 8 queens on the board.
- **Initial state -** No queen on the board
- **Actions -** Add a queen to any empty square
- **Transition model -** updated board
- **Goal test -** 8 queens on the board with none attacked





Start State



Goal State

Real-world Examples

Route finding problem - where we must go from location to location using links or transitions.

Traveling salesperson problem - find the shortest tour to visit each city exactly once.

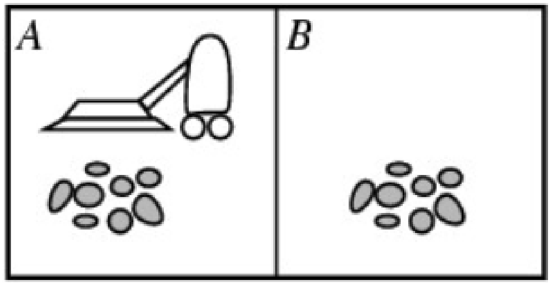
VLSI layout - position millions of components and connections on a chip to minimize area.

Robot navigation - This is a special case of route finding for robots with no specific routes or connections.

And, many more...

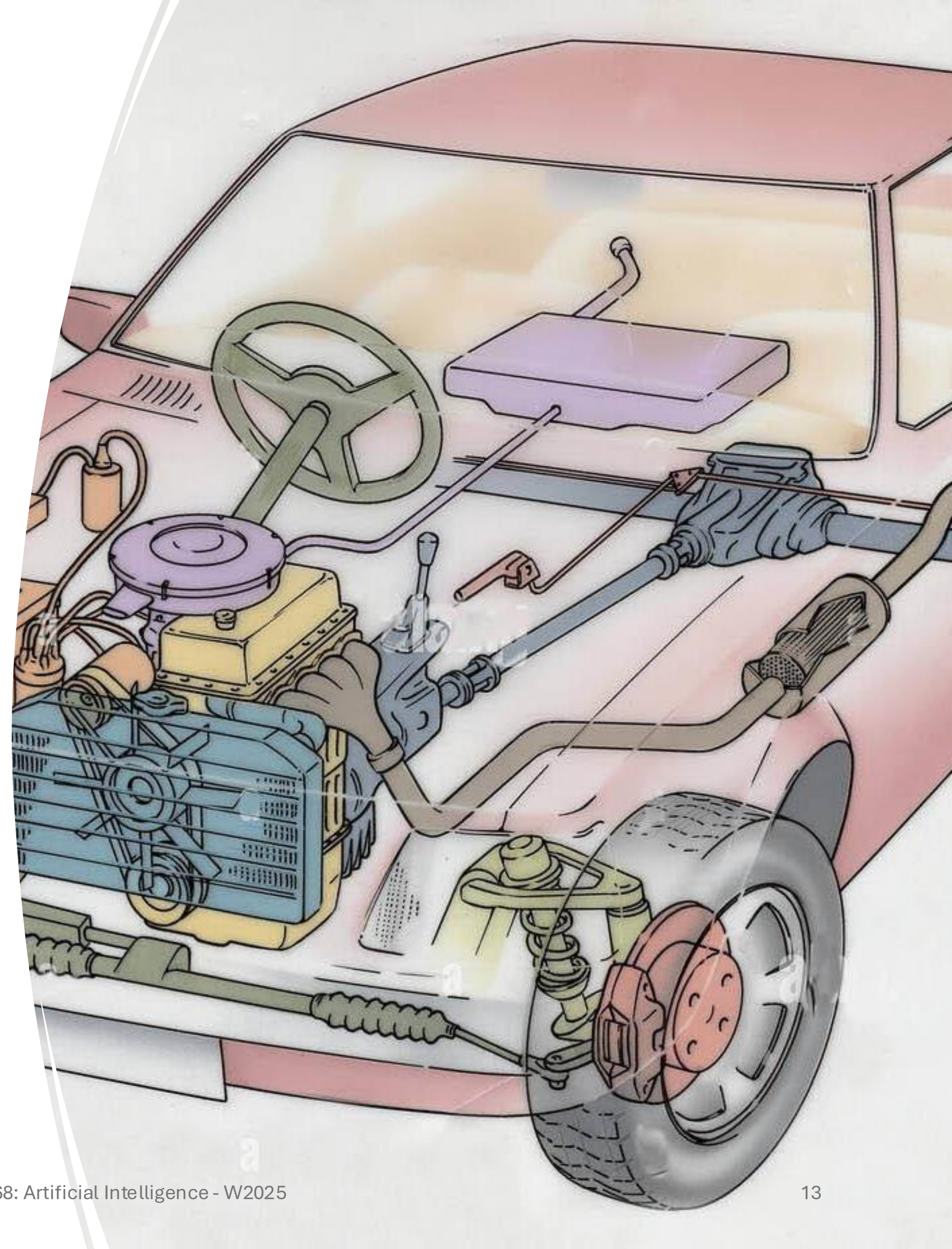
Problem Abstraction

- The real world is complex and has more details
- Irrelevant details should be removed from state space and actions, which is called abstraction.
- What’s the appropriate level of abstraction?
- Example: Vacuum-Cleaner



Abstraction

- Abstraction separates the purpose of an entity from its implementation or how it works
- Example in real life: a car (**we do not have to know how an engine works to drive a car**)

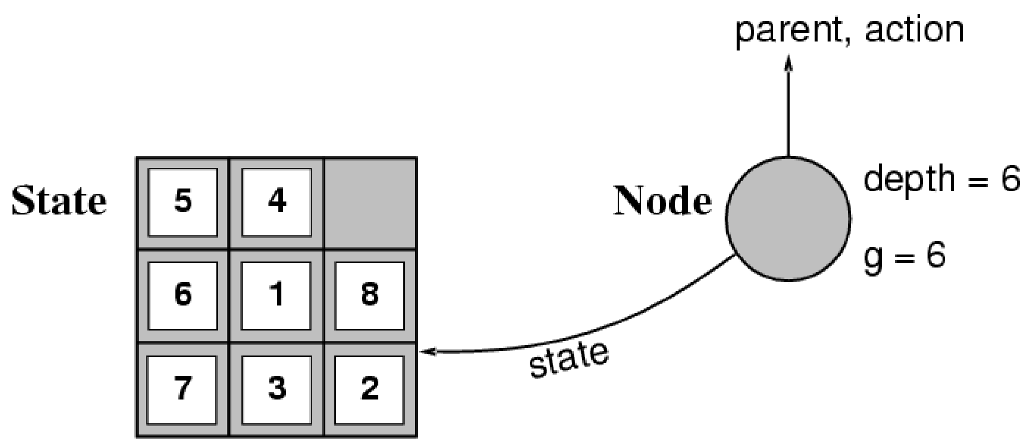


State space vs. Search space

- **State space** - a physical configuration
- **Search space** - an abstract configuration represented by a search tree or graph of possible solutions.
- **Search tree** - models the sequence of actions
  – Root, path cost, associated state: initial state
  – Branches: actions
  – Nodes: results from actions. A node has: parent, children, depth in the state space.
- **Expand** - A function that given a node, creates all children nodes

States vs. Nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree contains info such as: state, parent node, action, path cost *g(x)*, depth



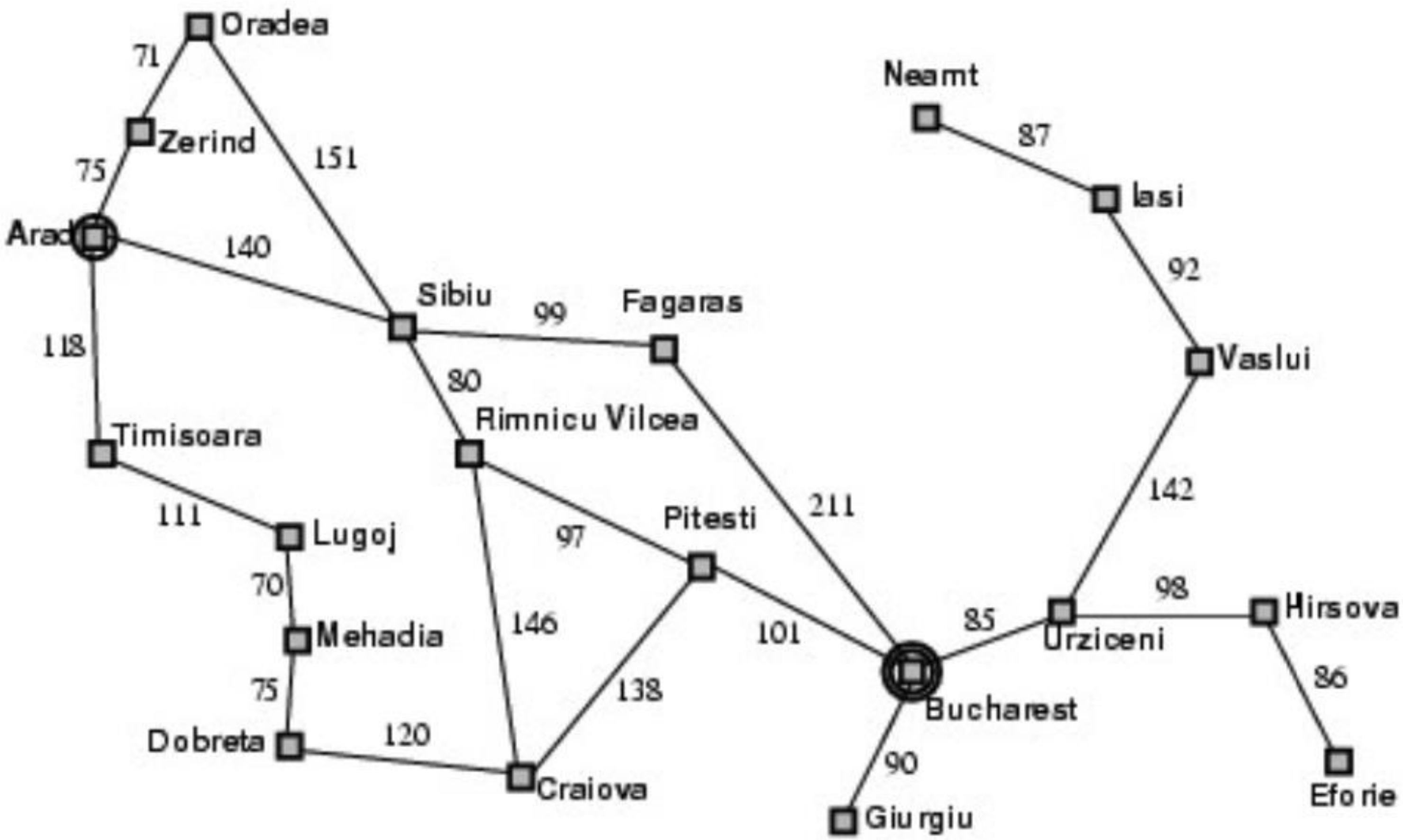
Search Space Regions

- The search space is divided into three regions
  - **Explored** (a.k.a. Closed List, Visited Set)
  - **Frontier** (a.k.a. Open List, the Fringe)
  - **Unexplored**.
- The essence of search is moving nodes from regions (3) to (2) to (1), and the essence of search strategy is deciding the order of such moves.

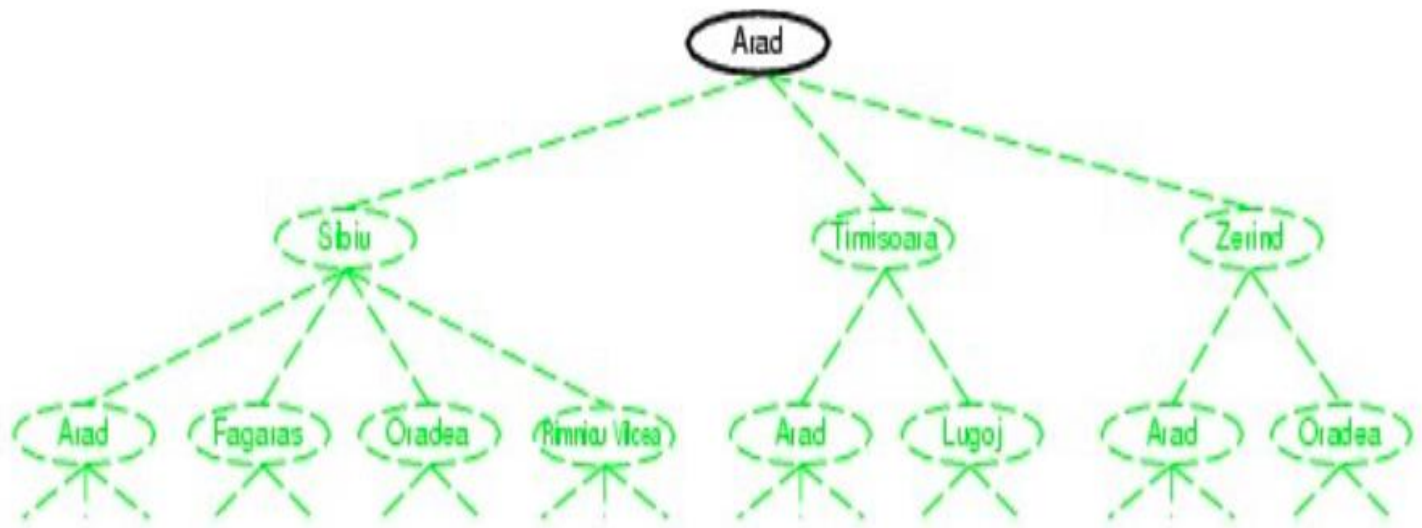
Searching for Solutions

- Search tree: generated by initial state and possible actions.
- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (expanding states)
  - the choice of which state to expand is determined by search strategy

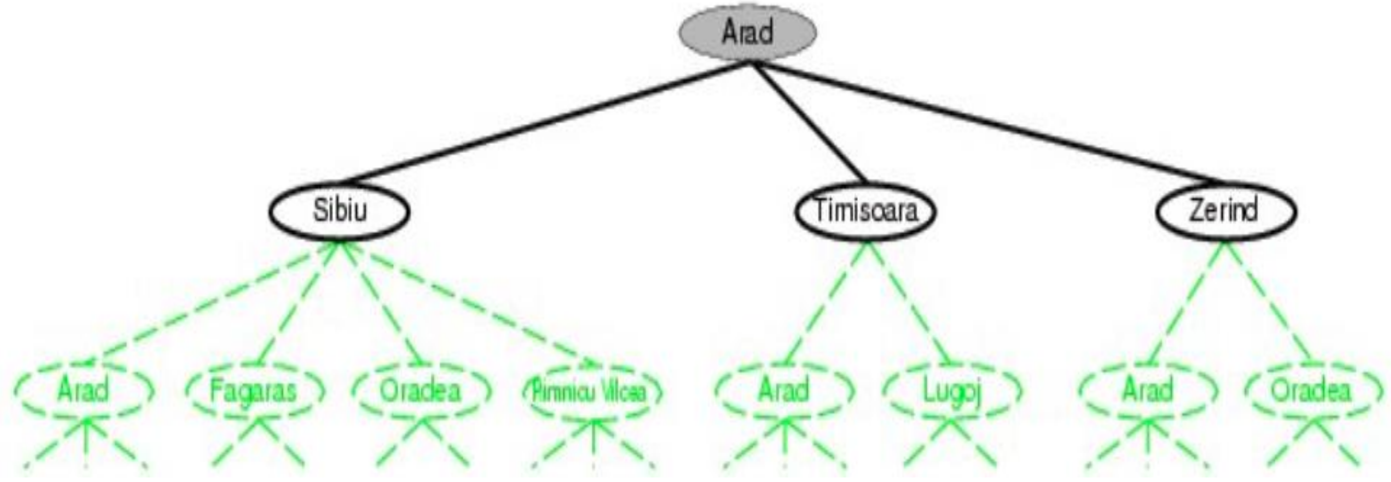
Road Map of Romania



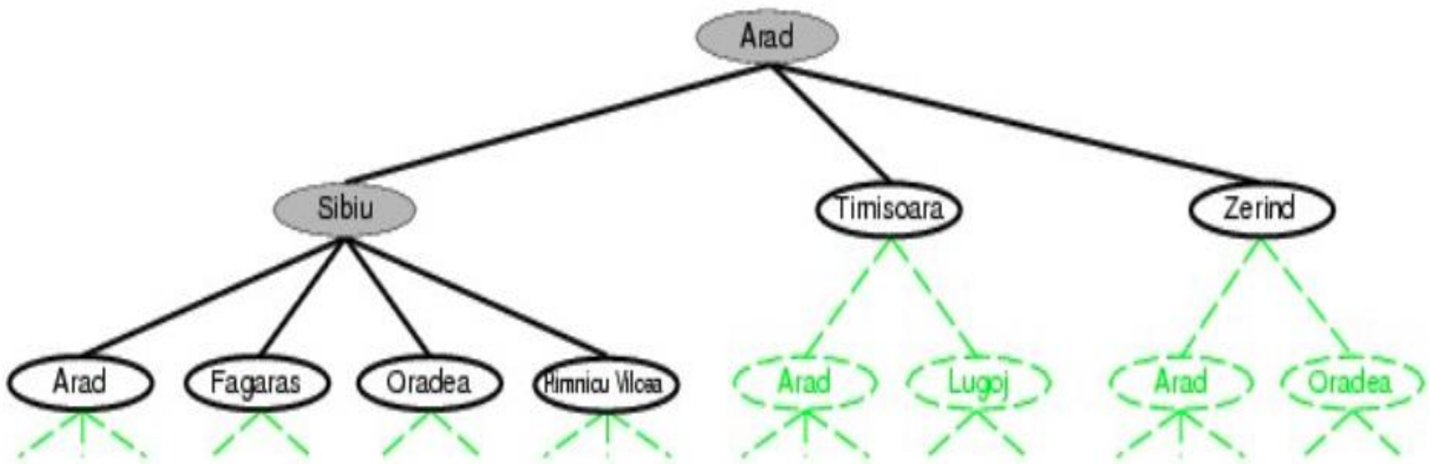
Tree Search Example



Tree Search Example



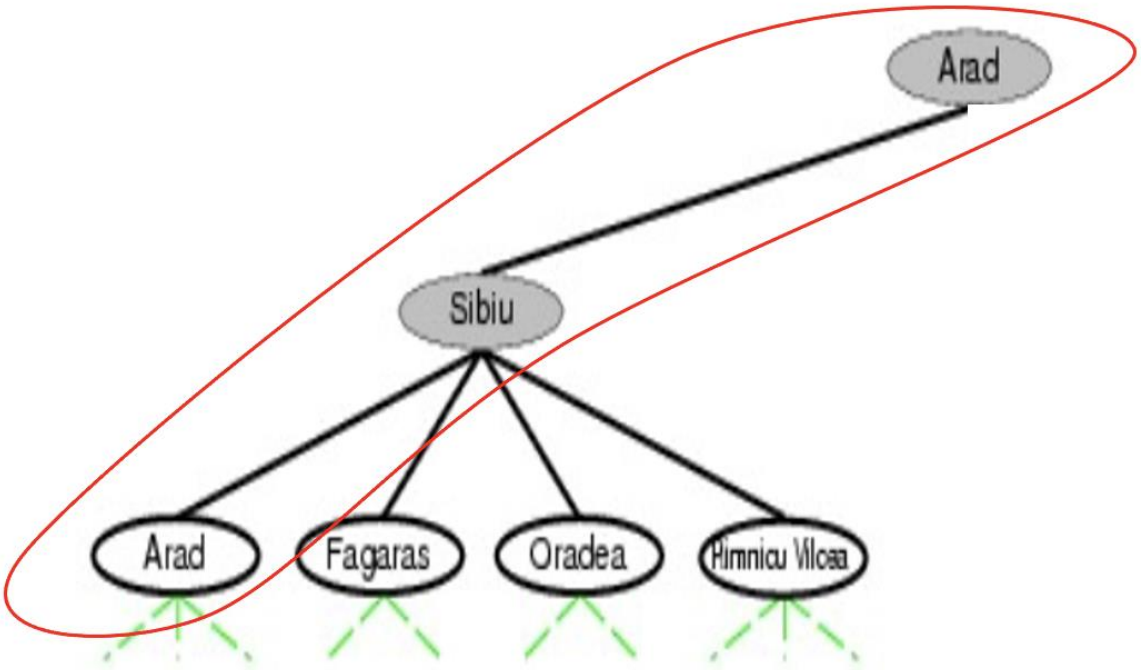
Tree Search Example



- Every state is evaluated: is it a goal state?

Avoiding Repeated States

- Failure to detect repeated states can turn a linear problem into an exponential one!



Search strategies

- A strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - Completeness - Does it always find a solution if one exists?
  - Time complexity - Number of nodes generated/expanded
  - Space complexity - Maximum number of nodes in memory
  - Optimality - Does it always find a least-cost solution?

Search strategies

- Time and space complexity are measured in terms of:
  - 'b' (**Branching Factor**): The maximum number of child nodes (actions) generated per state in the search tree.
  - 'd' (**Depth of Solution**): This is the distance from the root node (starting state) to the goal node (solution), measured in the number of edges in the path.
  - 'm' (**Maximum Depth of State Space**): This indicates the deepest level of the search tree, which could theoretically be infinite (∞) in some problems.

Types of Search Strategies

The two primary categories of search strategies differ based on the information available about the goal or state:

1. **Uninformed** Search (Blind Search): These strategies use only the goal's existence as information.
   - Pros**:** Simple to implement.
   - Cons**:** May explore many unnecessary states, leading to inefficiency.
2. **Informed** Search (Heuristic Search): These strategies use additional information (heuristics) to estimate the cost or distance to the goal.
   - Pros: More efficient in finding a solution.
   - Cons: Heuristic accuracy significantly impacts performance.