COMPX216/Y05337
Artificial Intelligence

Problem-solving agents

Credits: Prof. Eibe Frank (UoW)

Today: Problem-solving agents

- The agent and the environment
- Types of environments
- Problem-solving agents
- The problem-solving process
- Components of a search problem
- Example problems
- Search trees and search data structures
- Expanding nodes in a search tree
- The breadth-first search algorithm

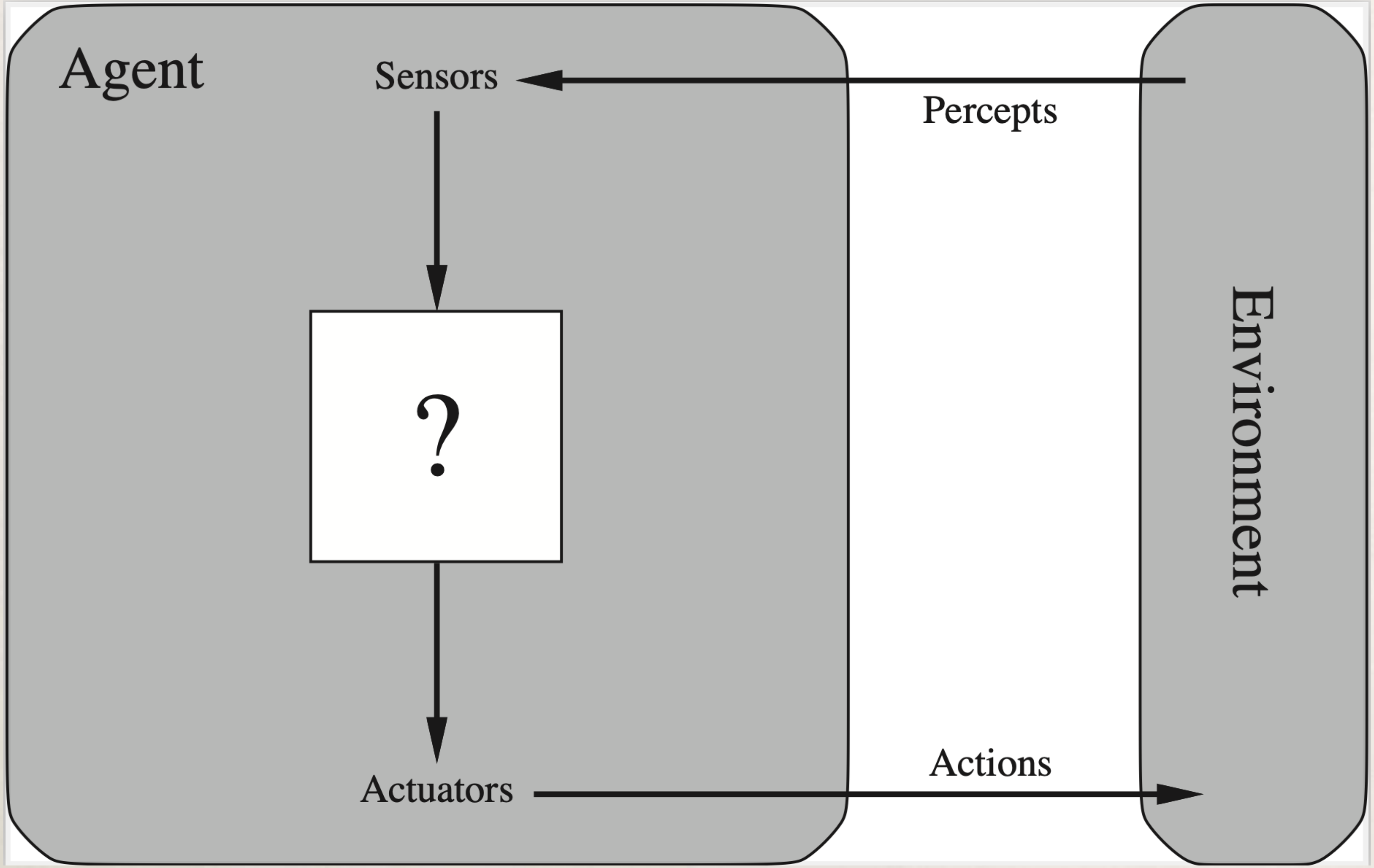
What is an Agent

- An agent exists in a specific environment
- The agent is the entity for which we want to make decisions and take actions
- In most cases, the environment defines a problem, while the agent gives solution

Agents and Environments

- An ***agent*** within an ***environment*** will take ***actions*** to achieve a particular ***goal***
- The agent ***perceives*** its environment somehow, and makes a ***decision*** about which action to take
- The agent takes its action(s), which ***changes*** the environment, and the process ***repeats***
- Artificial Intelligence is the algorithmic discipline of making these decisions more intelligent

The agent and the environment



Agents

- Perceives its environment through sensors
  - Robot: Camera, Laser, Sonar
  - Human: 5 Senses
  - Games: Keyboard Input / RAM / Game State
- Acts on its environment through Actuators
  - Robots: Wheels, Tracks, Arms
  - Humans: Limbs, Tools
  - Games: Predefined Rules / Actions
- May have some knowledge about environment

Agents

- Percept
  - Agent's perceptual inputs at any instant
  - "Agent's current belief of the world"
  - Environment's current State
- Percept Sequence
  - Complete history of everything the agent has ever perceived so far

Agents

- *In general,* an agent's choice of action at any given instant can depend on the entire percept sequence to date (what you did in past matters)
- Agent's behavior is given by the agent function that maps given percept sequence to an action
  - "Given what I have seen, take this action"

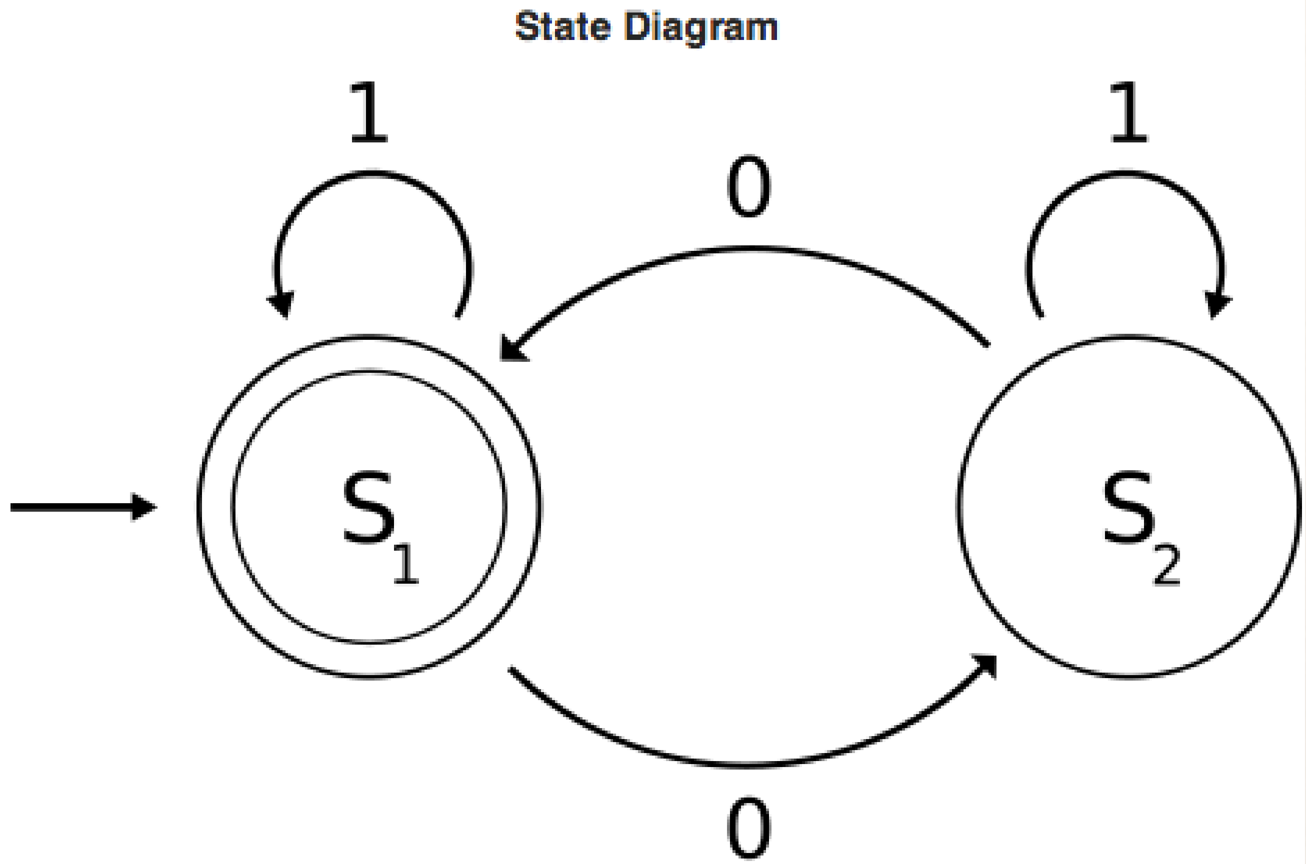
Agents

- An action taken by an agent in a given state transitions the state to another
- State Transition Function (STF)
  - Successor Function, Table, Graph
- State S, Action A
- S' = STF(S, A)

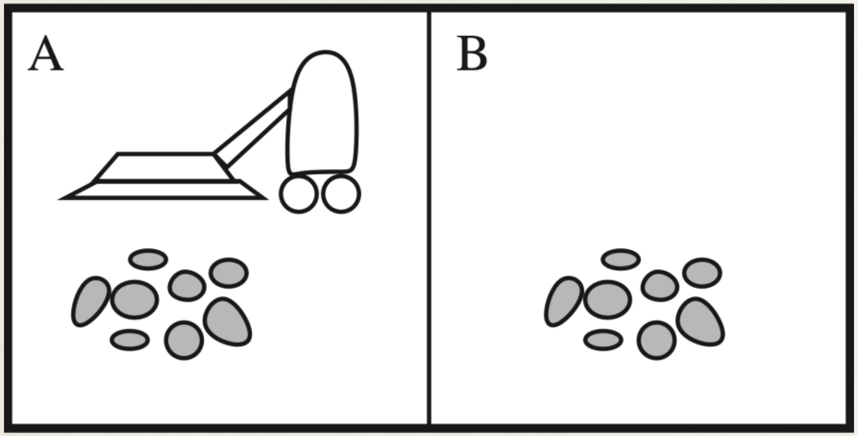
Agents

State Transition Table

Input State	1	0
S_1	S_1	S_2
S_2	S_2	S_1



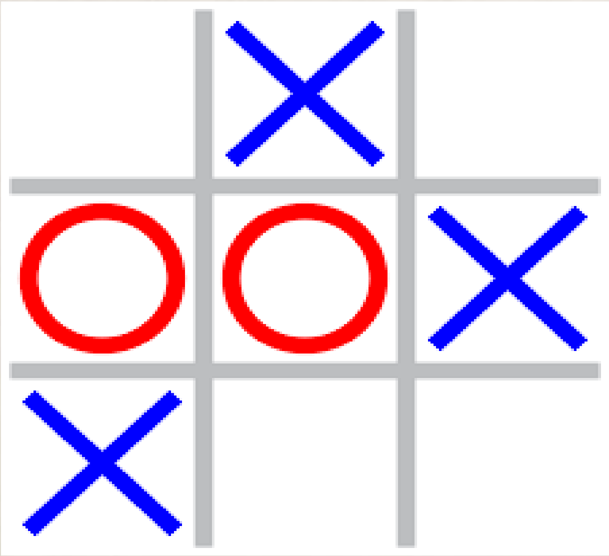
A very simple vacuum cleaner agent

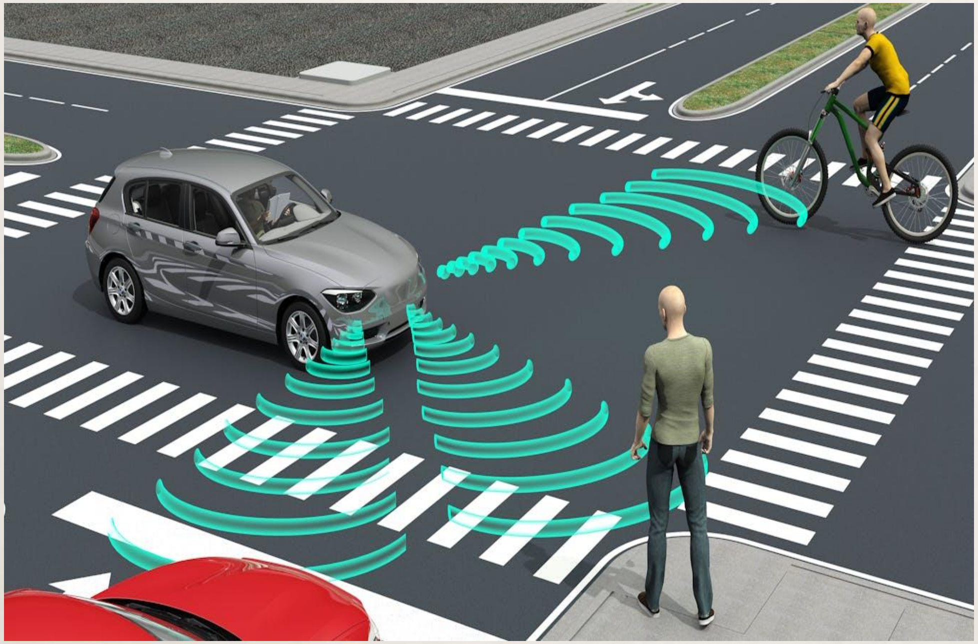


Percept	**Action**
[A, Clean]	Move right
[A, Dirty]	Suck dirt
[B, Clean]	Move left
[B, Dirty]	Suck dirt

Types of environments

- **Fully observable** vs. **partially observable**
  - Do the sensors provide access to the complete state of the environment?





Types of environments

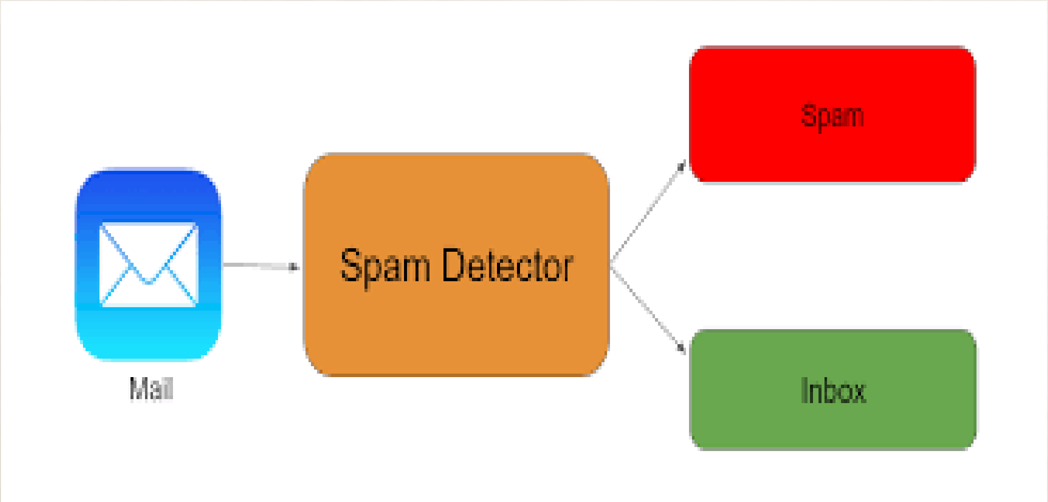
- **Single-agent** vs. **multi-agent**
  - Note that multi-agent environments can be competitive or cooperative

Types of environments

- **Deterministic** vs. **non-deterministic**
  - Note that partially observable environments may *appear* nondeterministic

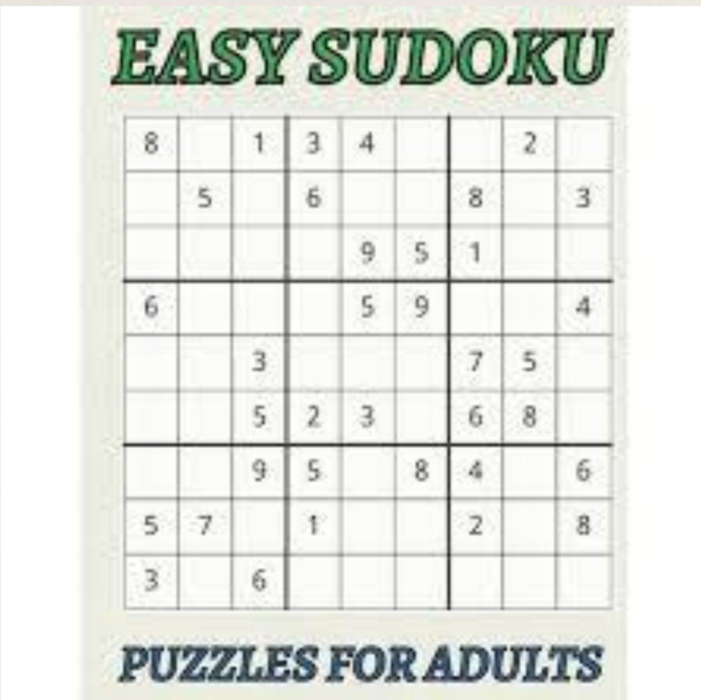
Types of environments

- **Episodic** vs. **sequential**
  - Does the outcome of an action only depend on the current state?



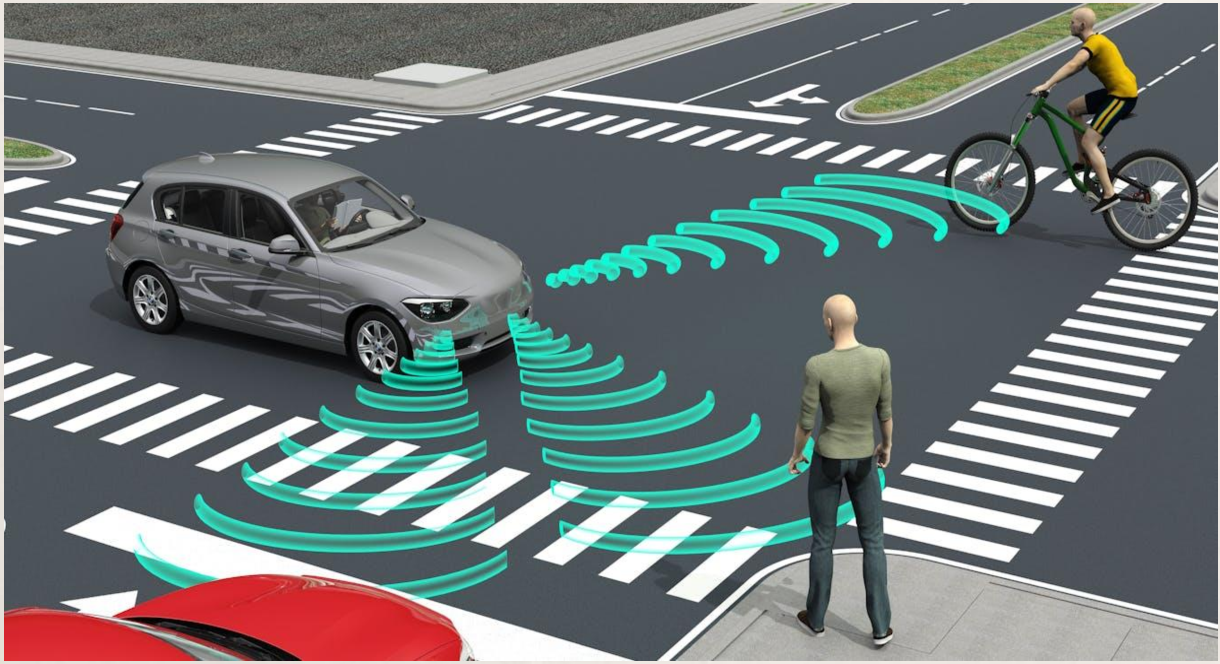
Types of environments

- **Static** vs. **dynamic**
  - Can the environment change while the agent is choosing an action?



Types of environments

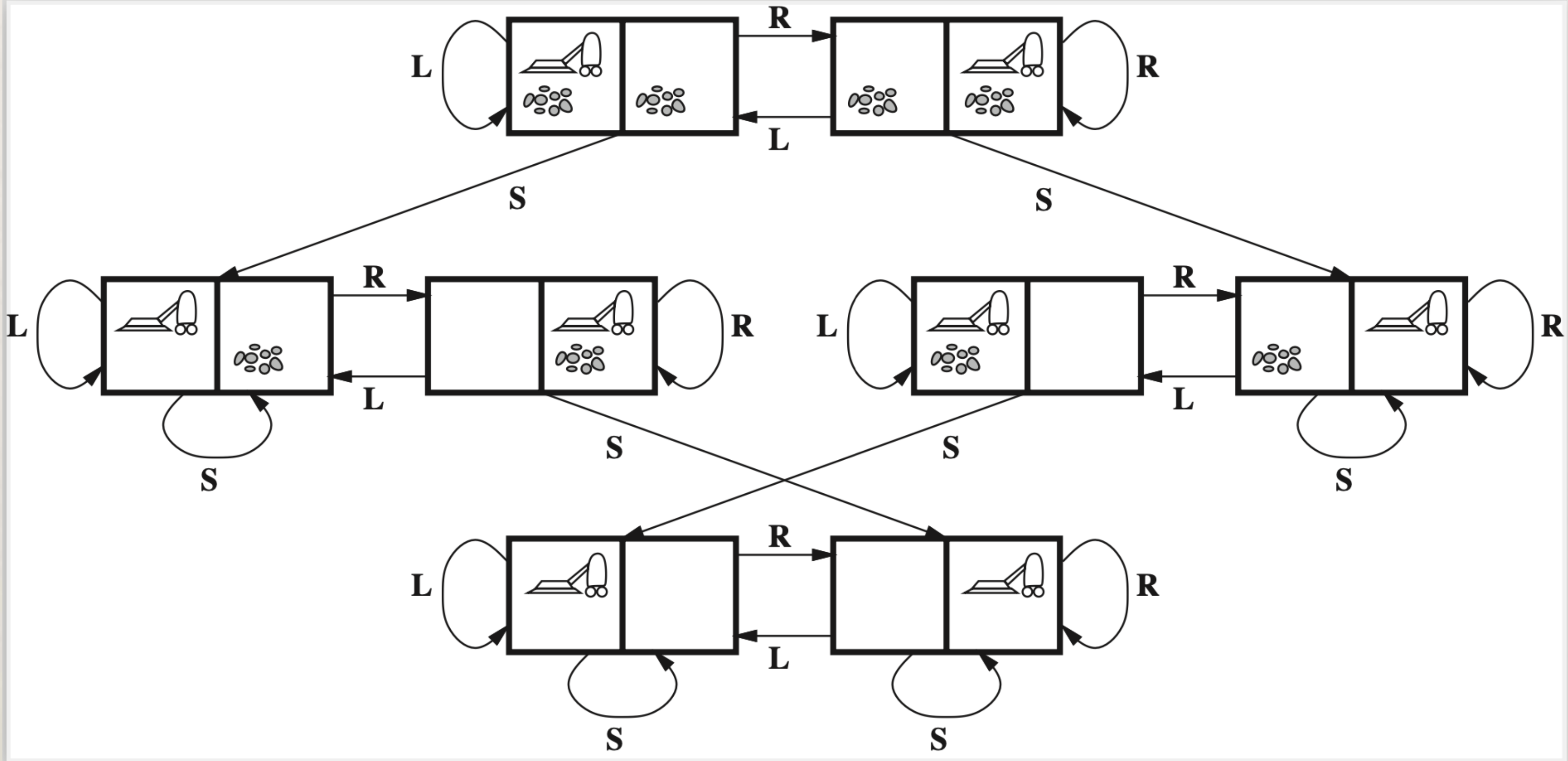
- **Discrete** vs. **continuous**
  - This distinction can apply to actions, percepts, states, and time



Problem-solving agents

- Correct action may not be obvious: we need to find a sequence of actions that form a **path** to the **goal state**
- We can do this by performing **search** in the **state space**
- **Problem-solving agents** use **atomic** state representations: the state of the world is considered indivisible
    - More complex **planning agents** are used for problems with **structured** or **factored** state representations
- The environments we consider are fully observable, single-agent, deterministic, episodic, static, and discrete
- We also assume that the effects of actions on the environment are known (i.e., its "physics" are known)

State-space graph for vacuum cleaner problem



The problem-solving process

- We can use the following four-step process to apply problem-solving agents to real-world problems
- **Goal formulation**: what is the goal of the search?
- **Problem formulation**: what is a suitable, sufficiently abstract model of relevant states and actions?
- **Search:** identify a sequence of actions that reach a goal state in the model, i.e., a path that is a **solution**
- **Execution:** perform the sequence of actions in the actual real-world environment
- We will look at algorithms for the third step

Components of a search problem

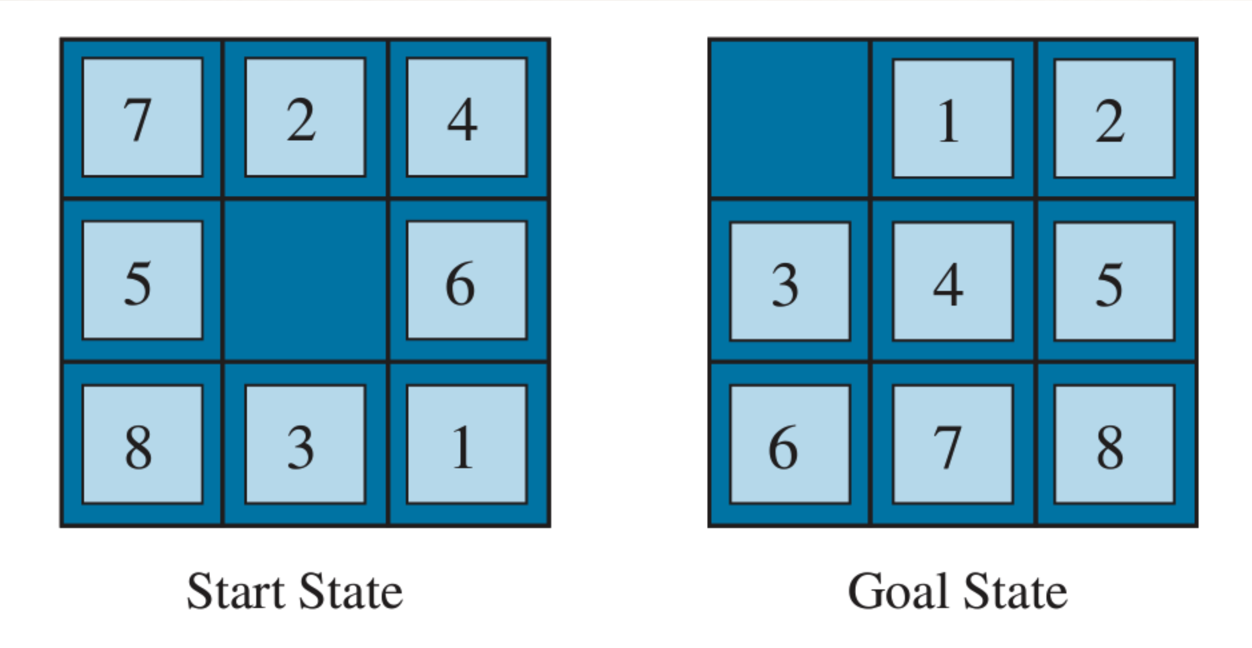
- **States**: the set of states the environment can be in
- **Initial state**: the starting point for the search
- **Goal states**: this set is implicitly defined by a Boolean function that tests whether a state is a goal state
- **Actions**: a function that returns the finite set of actions that are applicable in a given state
- **Transition model**: a function that defines the next state for a given state-action pair
- **Action cost function**: a function that defines a numeric cost of performing an action to move between two states
  - **Optimum solutions** are those with the lowest **path cost**

The vacuum world: problem definition

- **States**: a grid of cells, where each cell is either dirty or not, and the robot is in exactly one cell
- **Initial state**: any state in the state space
- **Goal states**: states in which all cells are clean
- **Actions**: *suck*, *move left*, *move right* (in a 2D world, we could add *move up* and *move down*)
- **Transition model**: a function that implements the transitions in the state-space graph from a few slides back
- **Action cost function**: each action has a cost of 1
    - This means the past cost is the number of actions it involves

The sliding-tile puzzle

- Most well-known sliding-tile puzzle: 8-puzzle



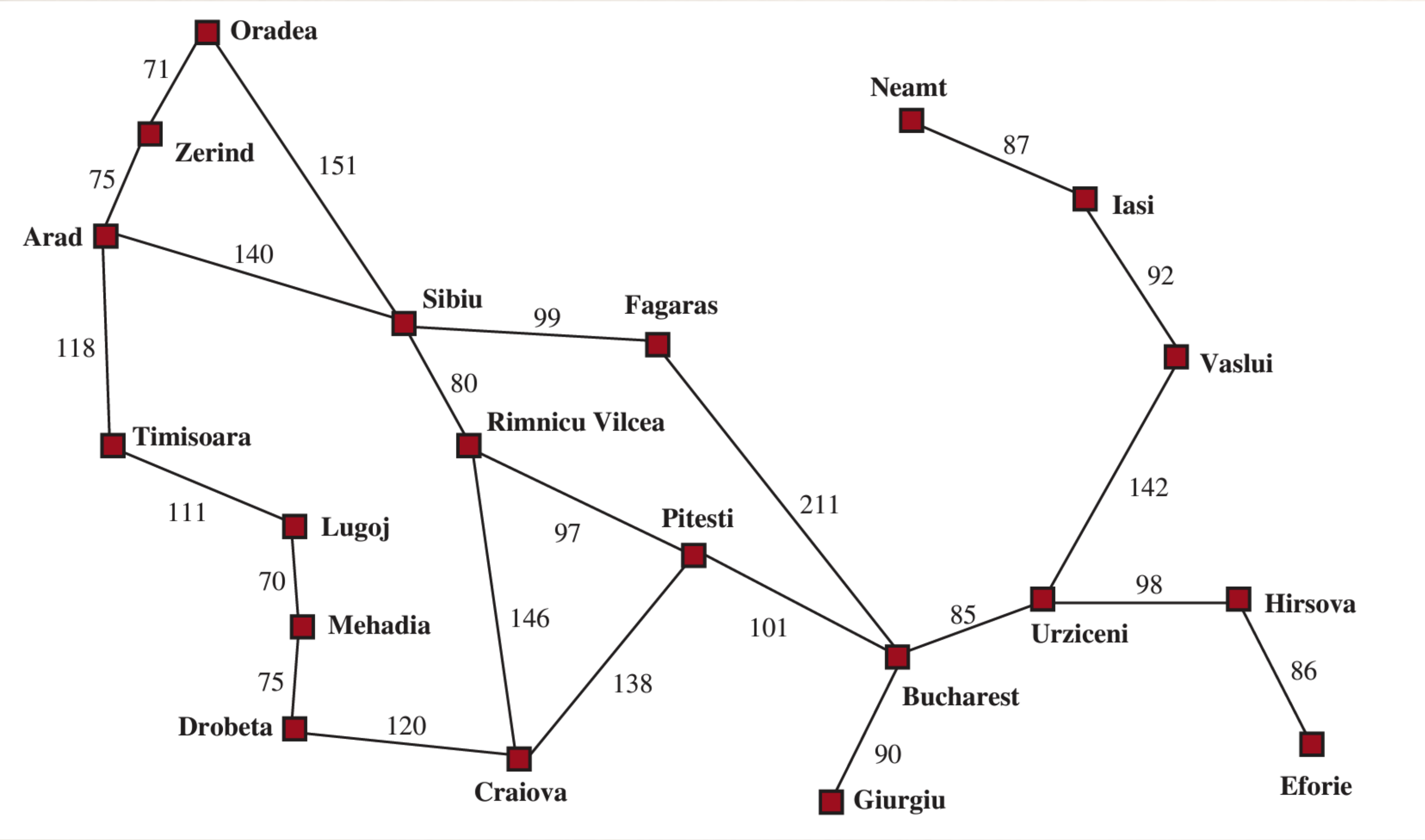
The sliding-tile puzzle: problem definition

- **States**: these specify the locations of all the tiles
- **Initial state**: any state in the state space
- **Goal states**: states where the numbers are in order
- **Actions**: moving the blank space *up, down, left,* or *right,* considering only applicable actions
- **Transition model**: a function that swaps the blank with an adjacent number based on the given action
- **Action cost function**: each action has a cost of 1

Knuth's conjecture & problem definition

- We can get any positive integer by starting with 4 and applying the square root, the floor function, or the factorial
- **States**: positive real numbers (note that this is infinite!)
- **Initial state**: the number 4
- **Goal state**: the desired positive integer
- **Actions**: apply the square root, floor function, or factorial (if applicable) to the current state
- **Transition model**: given by the definitions of the mathematical functions that are available
- **Action cost function**: each action has a cost of 1

Route finding (in Romania)

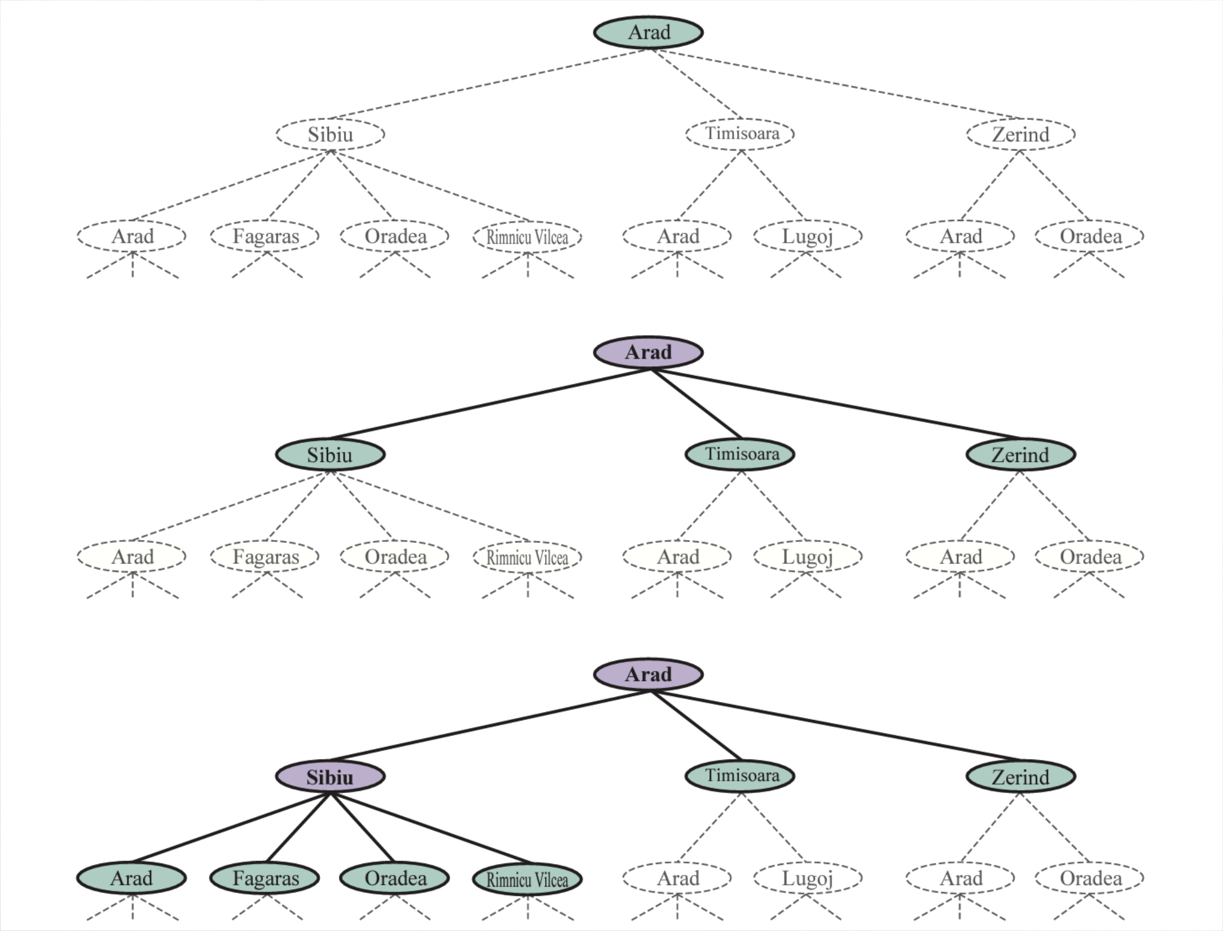


Route finding: problem definition

- **States**: locations on the map
- **Initial state**: any chosen location
- **Goal state**: the desired destination
- **Actions**: moves to adjacent cities on the map
- **Transition model:** given by the graph representing the map
- **Action cost function**: each action has a cost given by the distance between the two corresponding cities
- The optimum solution is the path with the shortest distance between the initial state and the goal state

Search trees: exploring paths in the state space

- **Search algorithms** construct a **search tree** that has **nodes** representing states on the paths generated so far



Search data structures

- `Node` objects represent nodes using five instance attributes:
  - `self.state`: a reference to an object representing a state
  - `self.parent`: a reference to the node in the tree from which this node was generated by performing an action
  - `self.action`: the action that was performed on the parent node's state to generate this node
  - `self.path_cost`: the total cost of the path from the initial state to this node
  - `self.depth`: the depth of the node in the search tree
- We also need to keep track of the **generated** but not yet **expanded** nodes (the so-called search **frontier**) using a **queue**
- Additionally, we should use a hashtable to check for states that have been **reached** (i.e., generated) previously

Node vs State

- State
    - Configuration of the environment
- Node
    - Bookkeeping data structure
    - Exists only within the search tree
    - Nodes are "on path" in the search tree

Expanding nodes

- To generate the search tree, we need a function that takes a node and uses the problem definition to generate children

function EXPAND($problem, node$) **yields** nodes
  $s \leftarrow node$.STATE
  for each $action$ **in** $problem$.ACTIONS(s) **do**
    $s' \leftarrow problem$.RESULT($s, action$)
    $cost \leftarrow node$.PATH-COST + $problem$.ACTION-COST($s, action, s'$)
    yield NODE(STATE=s', PARENT=$node$, ACTION=$action$, PATH-COST=$cost$)

- Note that this uses the yield keyword to indicate a resumable function (see generator functions Python!)

Search strategies

- Uninformed (blind) search
  - No information about states beyond problem description
  - Limited to children generation, goal test
- Informed (heuristic) search
  - Can guess which states are 'more promising'
  - Hopefully leads to faster search episodes

Which strategy to choose?

- Completeness
  - Is it guaranteed to find a solution if it exists?
- Optimality
  - Does it find the optimal solution?
- Time complexity
  - How long does it take to find a solution?
- Space complexity
  - How much memory is need to run the search?