Intelligent Agents

Chapter 2

Outline

◇ PAGE (Percepts, Actions, Goals, Environment)

◇ Environment types

◇ Agent types

PAGE

Must first specify the setting for intelligent agent design

Consider, e.g., the task of designing an automated taxi:

Percepts??

Actions??

Goals??

Environment??

PAGE

Must first specify the setting for intelligent agent design

Consider, e.g., the task of designing an automated taxi:

Percepts?? video, accelerometers, gauges, engine sensors, microphone, GPS, . . .

Actions?? steer, accelerate, brake, horn, speak/display, . . .

Goals?? safety, reach destination, maximize profits, obey laws, passenger comfort, . . .

Environment?? US urban streets, freeways, traffic, pedestrians, weather, customers, . . .

Environment types

	Chess	Backgammon	Taxi
Accessible??			
Deterministic??			
Episodic??			
Static??			
Discrete??			

Environment types

	Chess	Backgammon	Taxi
Accessible??	Yes	Yes	No
Deterministic??	Yes	No	No
Episodic??	No	No	No
Static??	Yes	Yes	No
Discrete??	Yes	Yes	No

The environment type largely determines the agent design

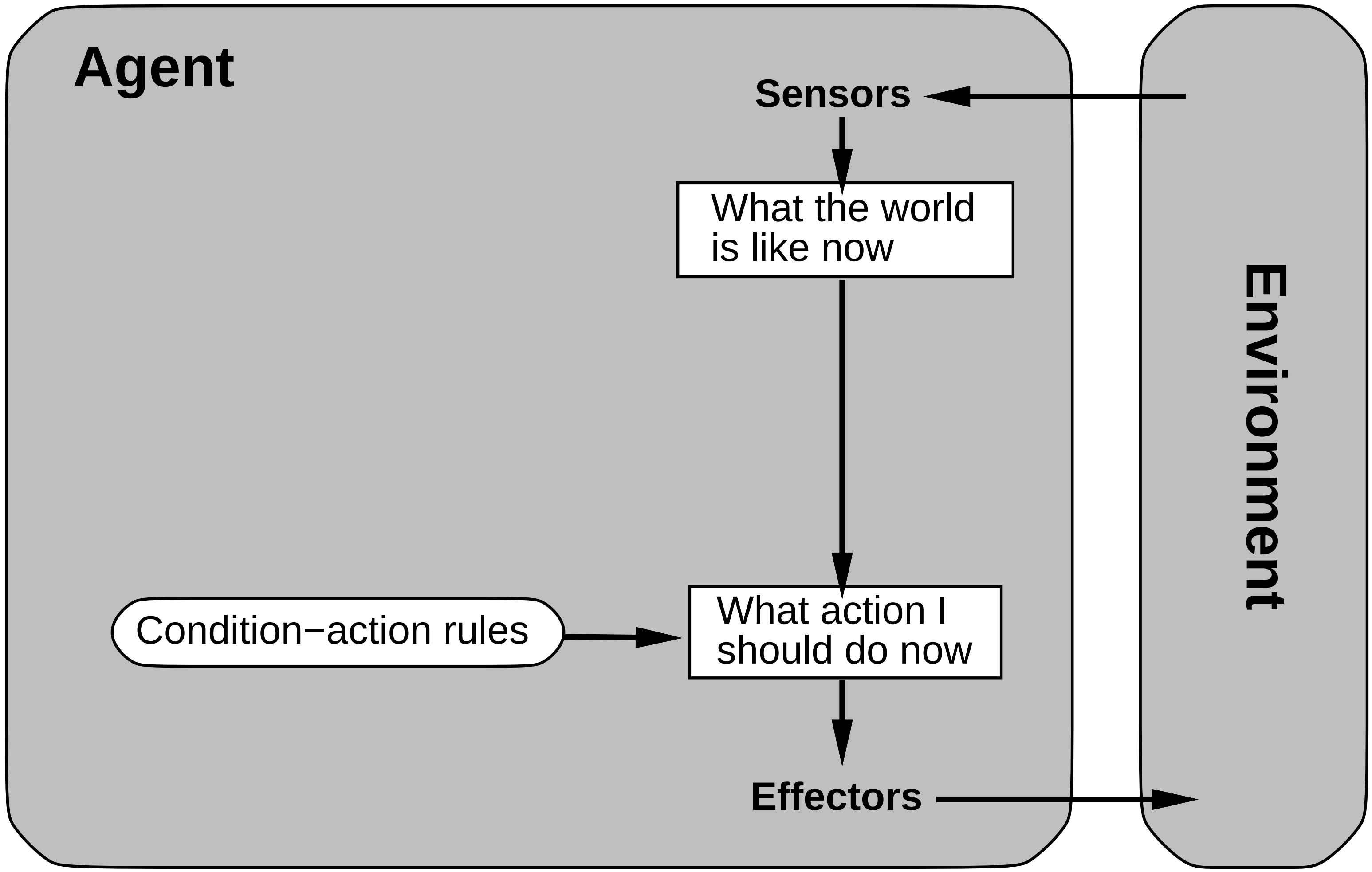
The real world is (of course) inaccessible, stochastic, sequential, dynamic, continuous

Agent types

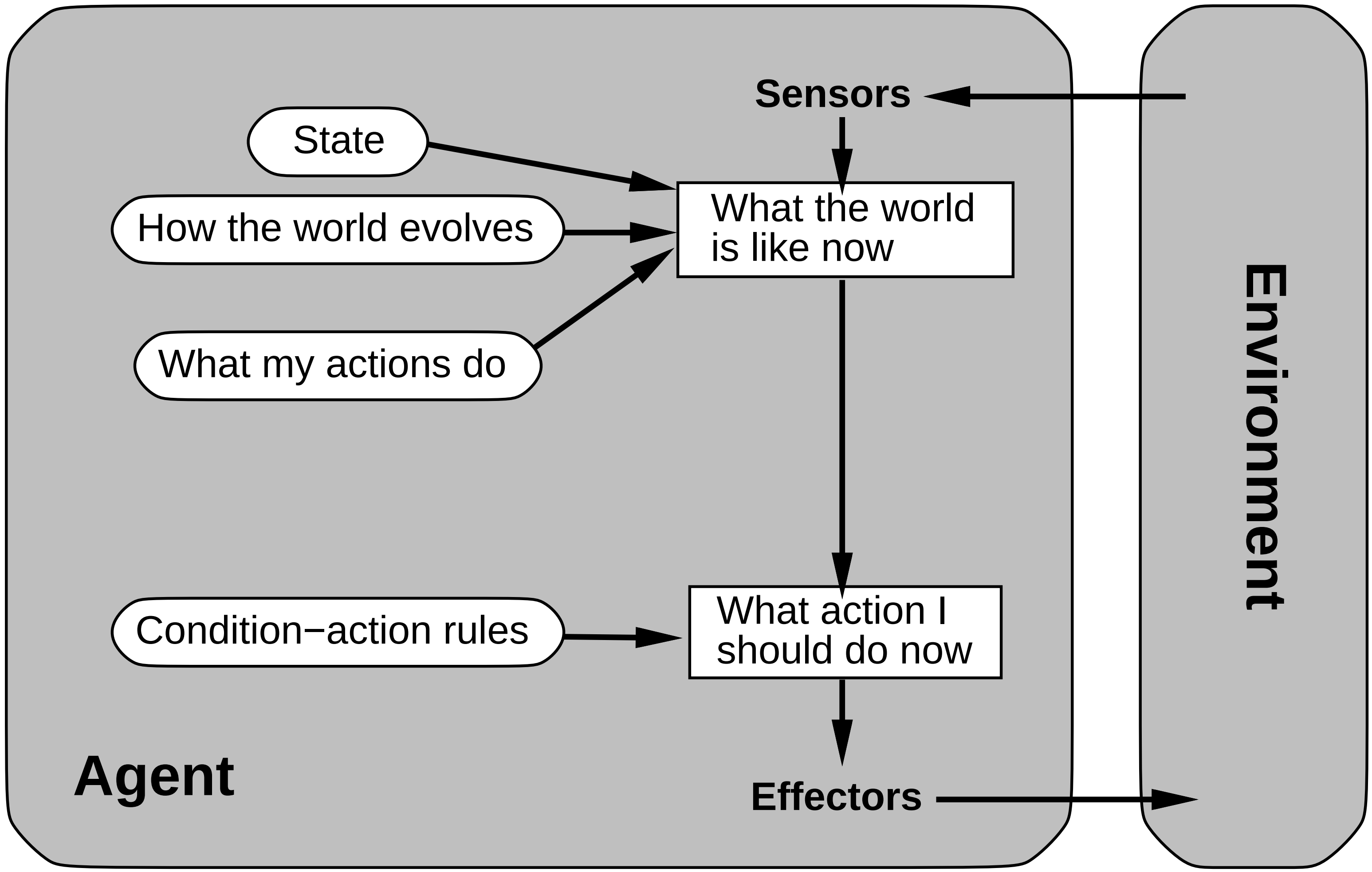
Four basic types in order of increasing generality:

- simple reflex agents
- reflex agents with state
- goal-based agents
- utility-based agents

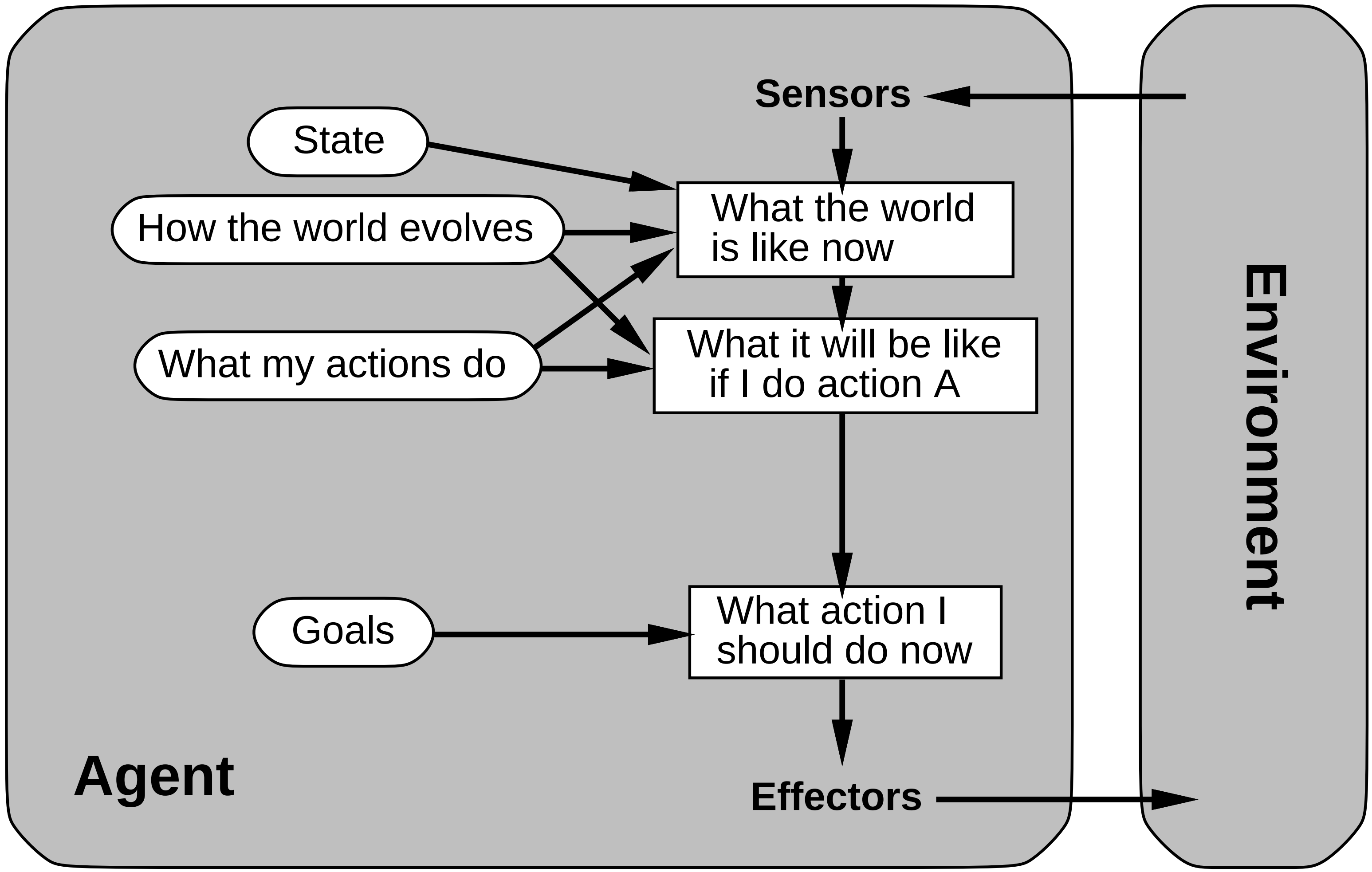
Simple reflex agents



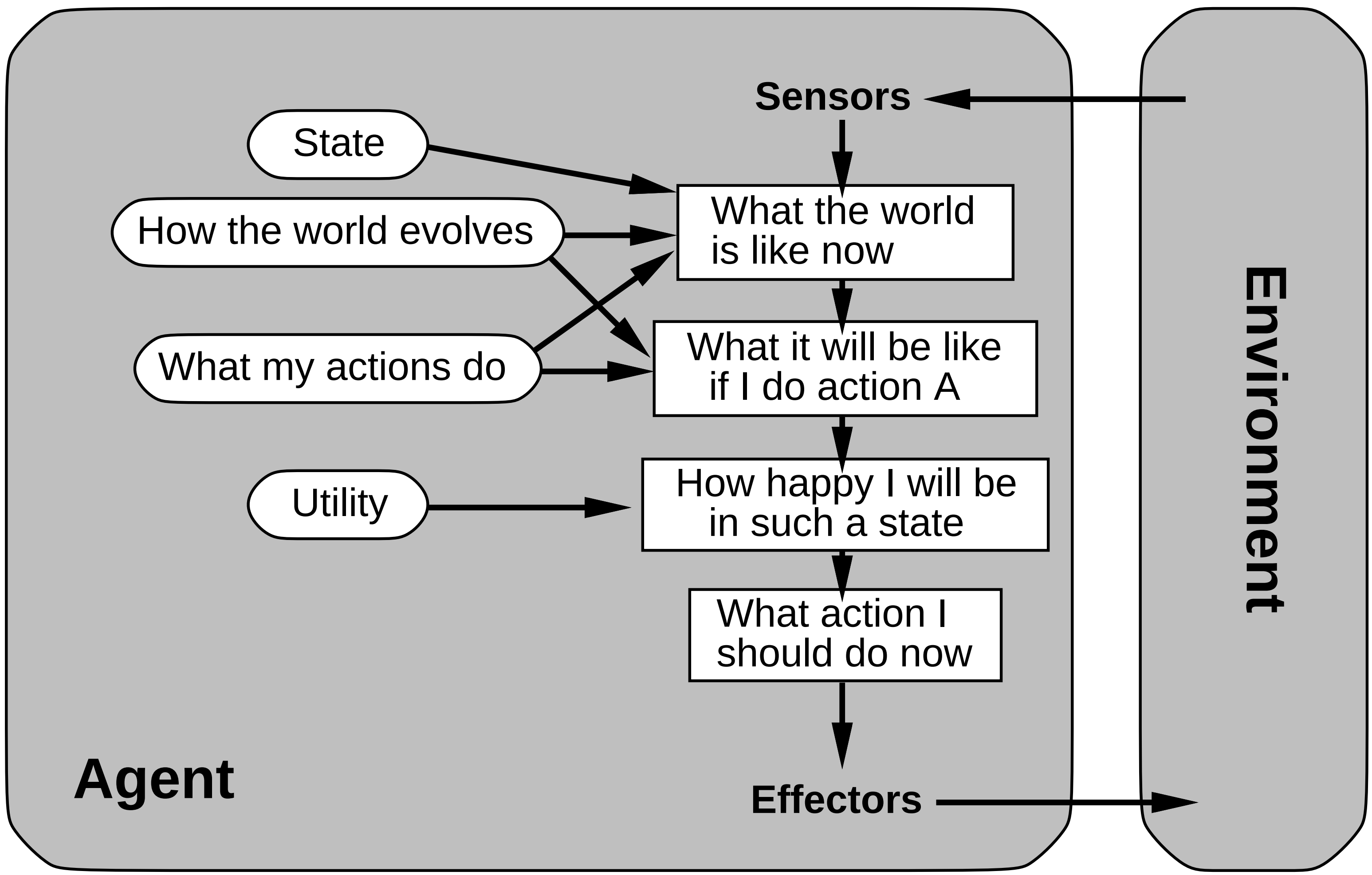
Reflex agents with state



Goal-based agents



Utility-based agents



Problem solving and search

Chapter 3

Outline

◇ Problem-solving agents

◇ Problem types

◇ Problem formulation

◇ Example problems

◇ Basic search algorithms

Problem-solving agents

Restricted form of general agent:

```
function SIMPLE-PROBLEM-SOLVING-AGENT( p) returns an action
   inputs: p, a percept
   static: s, an action sequence, initially empty
           state, some description of the current world state
           g, a goal, initially null
           problem, a problem formulation

   state ← UPDATE-STATE(state, p)
   if s is empty then
       g ← FORMULATE-GOAL(state)
       problem ← FORMULATE-PROBLEM(state, g)
       s ← SEARCH( problem)
   action ← RECOMMENDATION(s, state)
   s ← REMAINDER(s, state)
   return action
```

Note: this is *offline* problem solving.

Online problem solving involves acting without complete knowledge of the problem and solution.

Example: Romania

On holiday in Romania; currently in Arad.
Flight leaves tomorrow from Bucharest

Formulate goal:

be in Bucharest

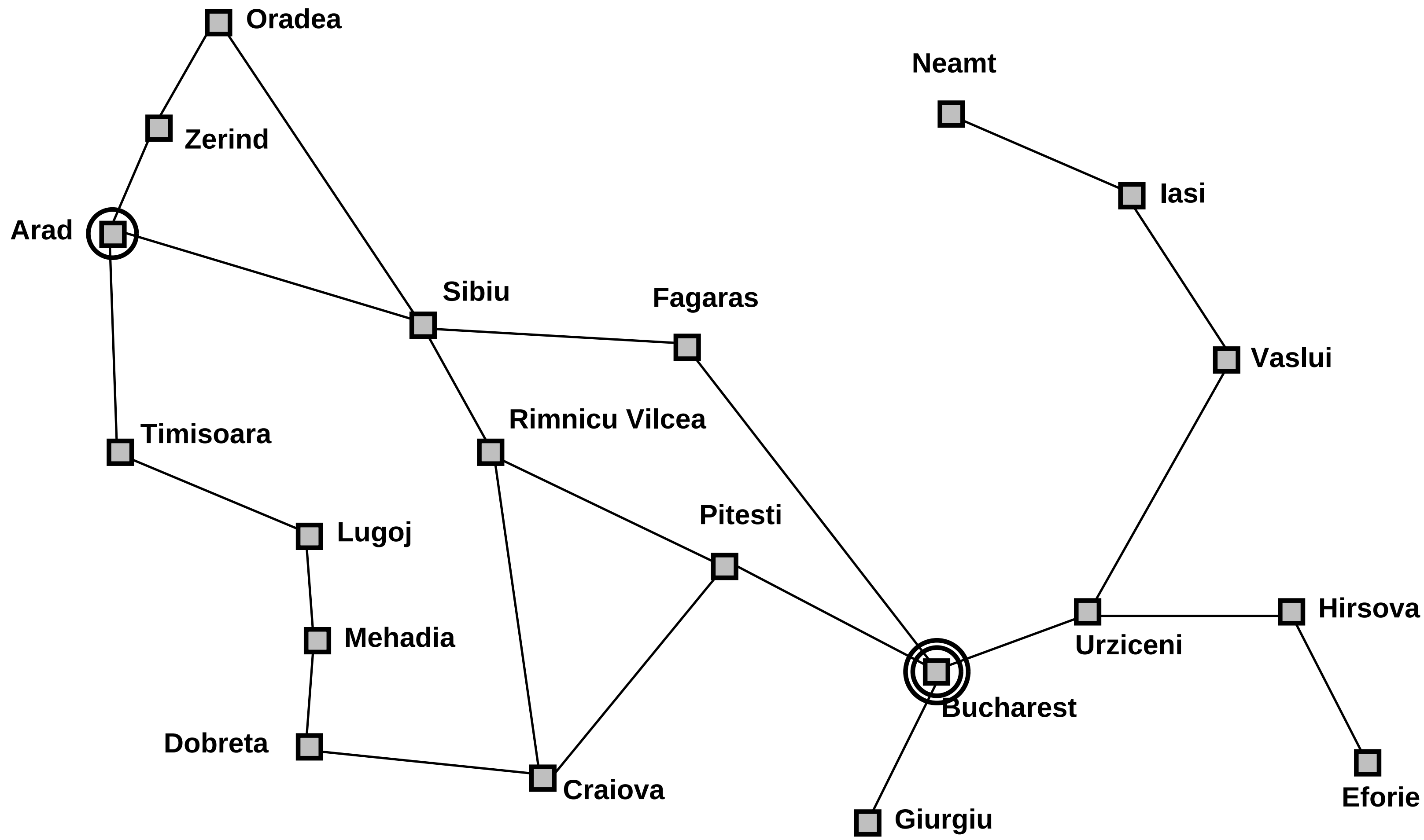
Formulate problem:

states: various cities
operators: drive between cities

Find solution:

sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Example: Romania



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")

Single-state problem formulation

A *problem* is defined by four items:

initial state   e.g., “at Arad”

operators (or *successor function* $S(x)$)

  e.g., Arad $\rightarrow$ Zerind   Arad $\rightarrow$ Sibiu   etc.

goal test, can be

  explicit, e.g., $x =$ “at Bucharest”

  implicit, e.g., $NoDirt(x)$

path cost (additive)

  e.g., sum of distances, number of operators executed, etc.

A *solution* is a sequence of operators
leading from the initial state to a goal state

Selecting a state space

Real world is absurdly complex

$\Rightarrow$ state space must be *abstracted* for problem solving

(Abstract) state = set of real states

(Abstract) operator = complex combination of real actions

e.g., "Arad $\rightarrow$ Zerind" represents a complex set of possible routes, detours, rest stops, etc.

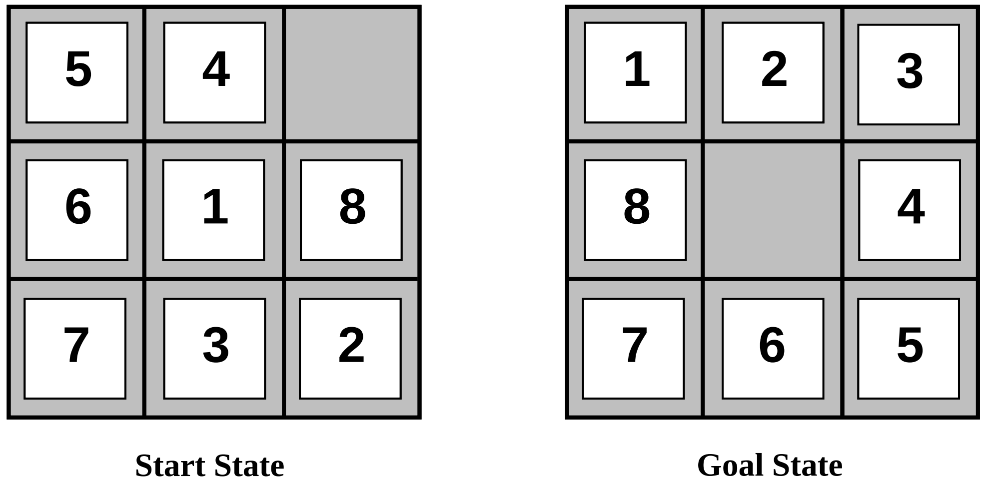
For guaranteed realizability, any real state "in Arad" must get to *some* real state "in Zerind"

(Abstract) solution =

set of real paths that are solutions in the real world

Each abstract action should be "easier" than the original problem!

Example: The 8-puzzle



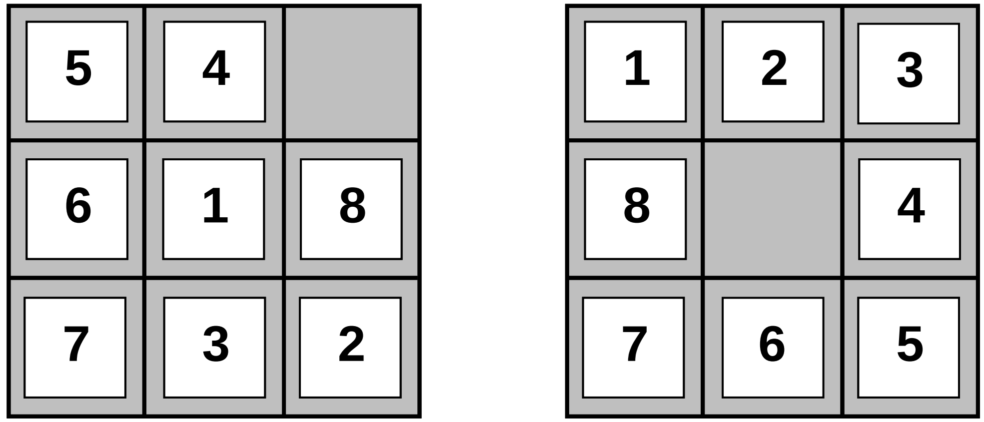
states??

operators??

goal test??

path cost??

Example: The 8-puzzle



Start State   Goal State

states??: integer locations of tiles (ignore intermediate positions)
operators??: move blank left, right, up, down (ignore unjamming etc.)
goal test??: = goal state (given)
path cost??: 1 per move

[Note: optimal solution of n-Puzzle family is NP-hard]

Example: service robot

states??:

operators??:

goal test??:

path cost??: