15381: Artificial Intelligence

Behrang Mohit

Intelligent Agents

Search

Some slides, graphics and ideas are borrowed or adapted from courses offered by Stuart Russell, Hwee Tou Ng, Rebecca Hwa and Milos Hauskrecht.

Four Approaches to AI

Thinking

Thinking Humanly | **Thinking Rationally**

Humanly ⟷ Rationally

Acting Humanly | **Acting Rationally**

Acting

2

Outline

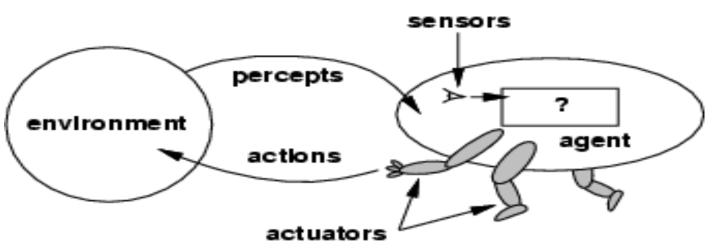
- Rational Agents:
  - Agents and environments
  - Rationality
  - Environment types
  - Agent types
- Problem solving with **search**

3

Intelligent Agents

Agents

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators



5

Sample agents

- Human agent:
  - Sensors: eyes, ears, and other organs
  - Actuators: Hands, legs, mouth, and other body parts
- Robotic agent:
  - Sensors: cameras and infrared range finders
  - Actuators: various motors

6

Agents and environments

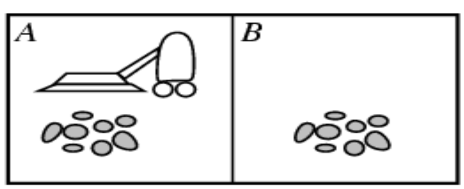
- The agent function maps from percept histories to actions:

$$[f: \mathcal{P}^* \rightarrow \mathcal{A}]$$

- The agent program runs on the physical architecture to produce *f*
  - agent = architecture + program

7

Vacuum-cleaner world

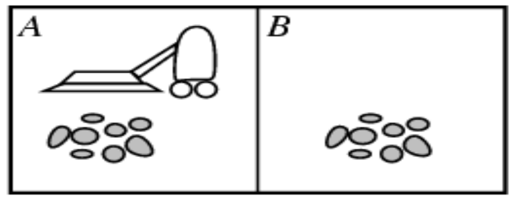


- Percepts: location and contents, e.g., [A,Dirty]
- Actions: *Left*, *Right*, *Suck*, *NoOp*

8

A vacuum-cleaner agent

Percept Sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
..	..
[A, Clean], [A, Dirty]	Suck
..	..
[A, Clean], [A, Clean], [A, Dirty]	Suck



Rational Agent does the right thing!

- An agent should strive to ***do the right thing***, based on what it can perceive and the actions it can perform. The *right* action is the one that will cause the agent to be most successful

Measuring the success

Performance measure: An objective criterion for success of an agent's behavior

- E.g., performance measure of a vacuum-cleaner agent
  - amount of dirt cleaned up,
  - amount of time taken
  - amount of electricity consumed
  - amount of noise generated

Rational agents

- For each possible percept sequence, a rational agent should select an action that **is expected to maximize its performance measure**, given the evidence provided by the *percept sequence* and whatever built-in knowledge the agent has.

Rational agents

- Rationality is distinct from omniscience (all-knowing with infinite knowledge)
- Agents can perform actions in order to modify future percepts so as to obtain useful information (information gathering, exploration)
- An agent is autonomous if its behavior is determined by its own experience (with ability to learn and adapt)

13

Task Environment

- PEAS: **P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors
- Must first specify the setting for intelligent agent design

14

Example: Automated Taxi Driver

- Performance measure ??
- Environment ??
- Actuators ??
- Sensors ??

15

Example: Automated Taxi Driver

- Performance measure: Safe, legal, comfortable, max profit
- Environment: Roads, other traffic, pedestrians, customers
- Actuators: Steering wheel, accelerator, brake, signal, horn
- Sensors: Cameras, speedometer, GPS, odometer, engine sensors, keyboard

16

Example: Medical Diagnosis system

- Agent: Medical diagnosis system
- Performance measure: Healthy patient, minimize costs, lawsuits
- Environment: Patient, hospital, staff
- Actuators: Screen display (questions, tests, diagnoses, treatments, referrals)
- Sensors: Keyboard (entry of symptoms, findings, patient's answers)

17

Example: Interactive English Tutor

- Agent: Interactive English tutor
- Performance measure: Maximize student's score on test
- Environment: Set of students
- Actuators: Screen display (exercises, suggestions, corrections)
- Sensors: Keyboard

18

Environment types

- Fully observable (vs. partially observable): An agent's sensors give it access to the complete state of the environment at each point in time.
- Deterministic (vs. stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent.
- Episodic (vs. sequential): The agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action)

19

Environment types

- Static (vs. dynamic): The environment is unchanged while an agent is deliberating.
- Discrete (vs. continuous): A limited number of distinct, clearly defined percepts and actions.
- Single agent (vs. multiagent): An agent operating by itself in an environment.

20

Environment types

Observable	Observable	Agent	Deterministic	Episodic	Static	Discrete
Crossword puzzle						
Taxi driving						
Medical diagnosis						
Interactive Tutor						

- The environment type largely determines the agent design

Environment types

Observable	Observable	Agent	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequen.	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequen.	Dyn.	Contin.
Medical diagnosis	Partially	Single	Stochastic	Sequen.	Dyn.	Contin.
Interactive Tutor	Partially	Multi	Stochastic	Sequen.	Dyn.	Discrete

- The **real world** is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent

Goal: Rational agent functions

- An agent is completely specified by the agent function mapping percept sequences to actions

- Aim: find a way to implement the rational agent function concisely

23

Table-lookup agent

- For every percept, look up the action table (fully specified).

- Drawbacks:
  - Huge table
  - Take a long time to build the table
  - No autonomy
  - Even with learning, need a long time to learn the table entries

24

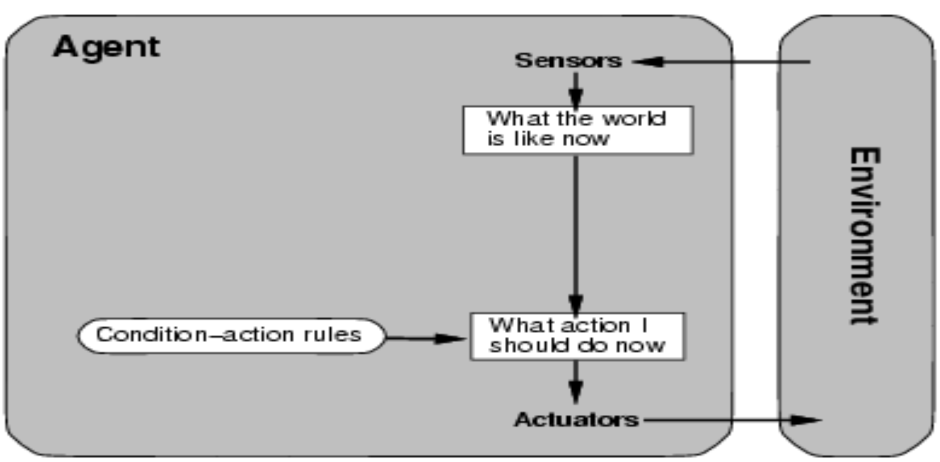
Agent types

- Four basic types in order of increasing generality:

- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

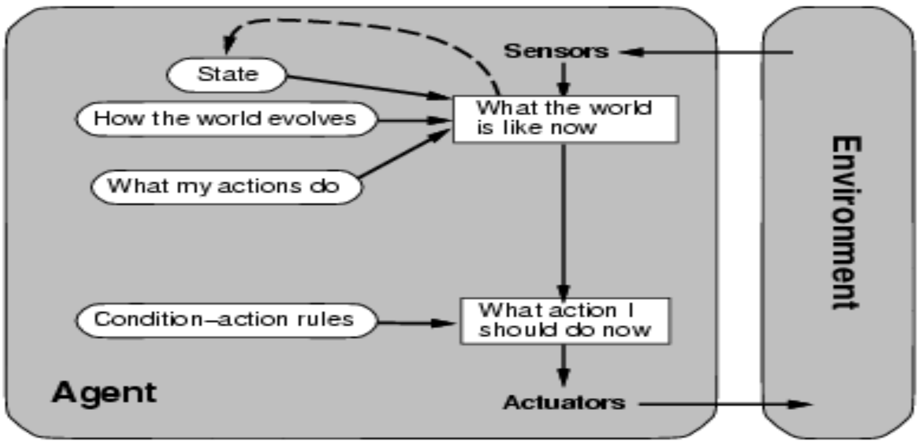
25

Simple reflex agents



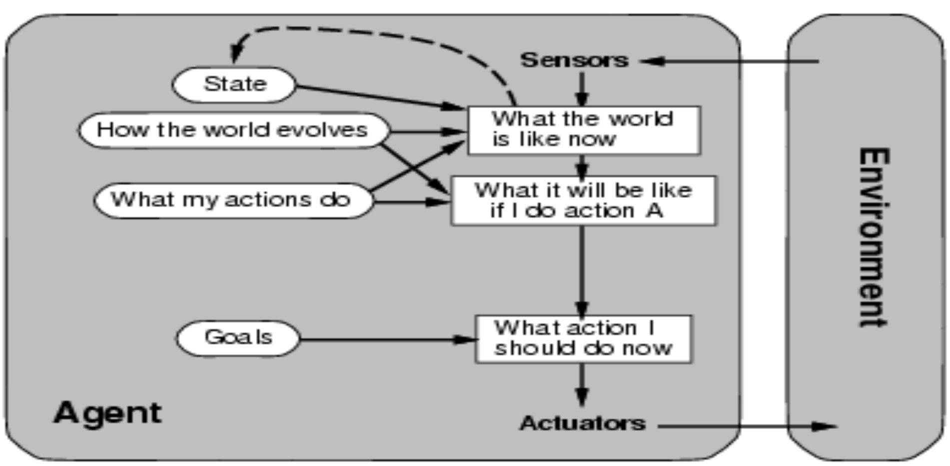
26

Model-based reflex agents



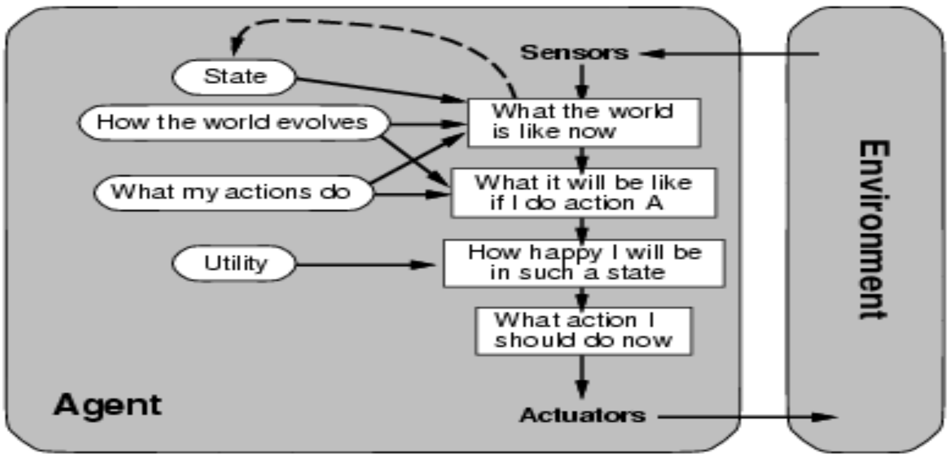
27

Goal-based agents



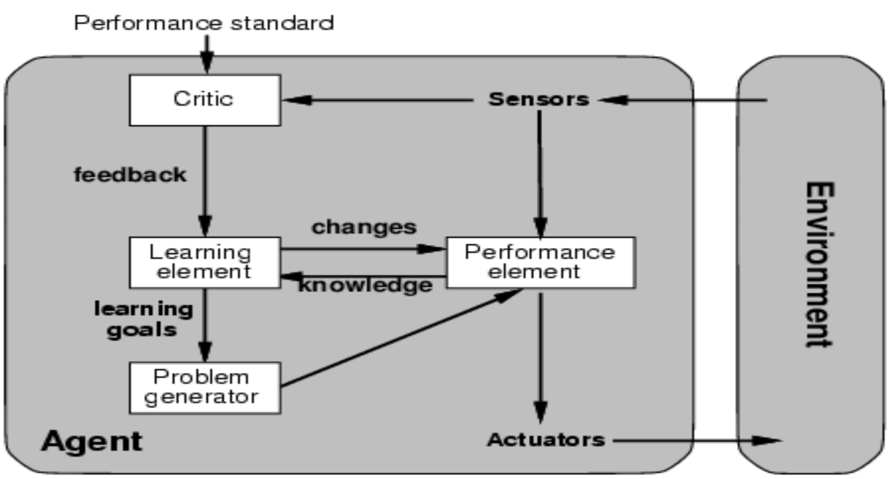
28

Utility-based agents



29

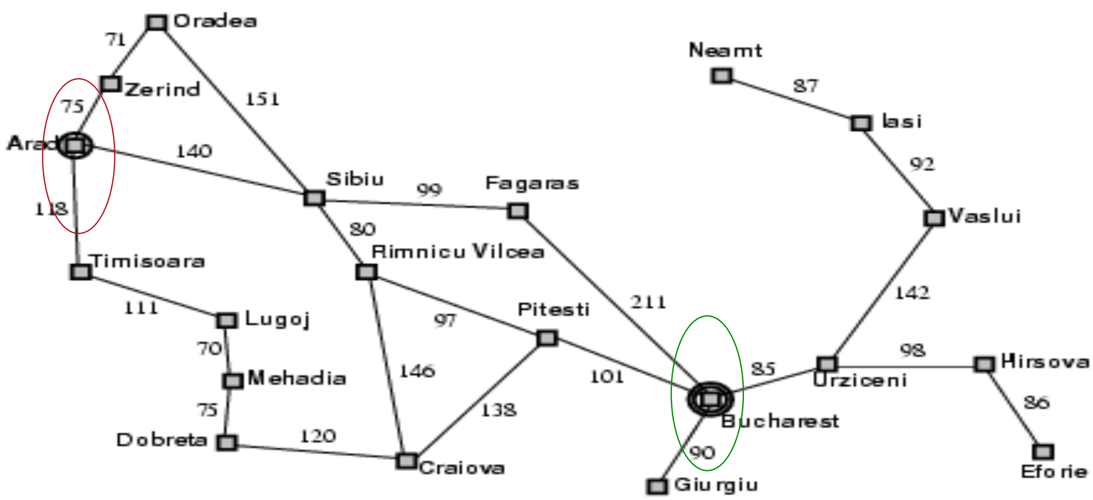
Learning agents



30

Solving Problems with Search

Example Problem: Romania Trip



32

Problem-solving agents

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

   state ← UPDATE-STATE(state, percept)
   if seq is empty then do
       goal ← FORMULATE-GOAL(state)
       problem ← FORMULATE-PROBLEM(state, goal)
       seq ← SEARCH( problem)
   action ← FIRST(seq)
   seq ← REST(seq)
   return action
```

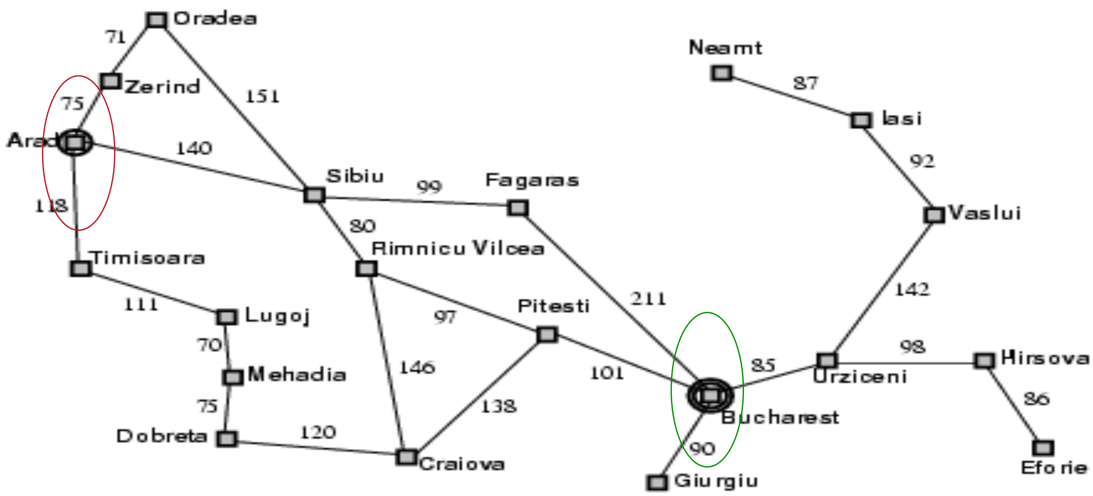
33

Example: Romania Trip

- Formulate goal:
  - be in Bucharest
- Formulate problem:
  - states: various cities
  - actions: drive between cities
- Find solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

34

Example: Romania Trip



35

Problem formulation

A problem is defined by four items:

- Initial state
- Actions or successor function
- Goal test
- Path cost

36

Problem formulation

A problem is defined by four items:

- Initial state e.g., "at Arad”
- Actions or successor function
- Goal test
- Path cost

37

Problem formulation

A problem is defined by four items:

- Initial state
- Actions or successor function
    - *S(x)* = set of action–state pairs
      e.g., *S(Arad)* = {<*Arad* → *Zerind, Zerind*>, … }
- Goal test
- Path cost

38

Problem formulation

A problem is defined by four items:

- Initial state
- Actions or successor function
- Goal test can be
  - explicit, e.g., x = "at Bucharest"
  - implicit, e.g., *Checkmate(x)*

- Path cost

39

Problem formulation

A problem is defined by four items:

- Initial state
- Actions or successor function
- Goal test
- Path cost
  - e.g., sum of distances, number of actions executed, etc.
  - $c(x,a,y)$ is the step cost, assumed to be ≥ 0

40

Problem formulation

A problem is defined by four items:

- Initial state
- Actions or successor function
- Goal test
- Path cost

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

41

Selecting a state space

- Real world is absurdly complex
  → state space must be abstracted for problem solving

- (Abstract) state = set of real states

42

Selecting a state space

- (Abstract) action = complex combination of real actions
  - e.g., "Arad → 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”

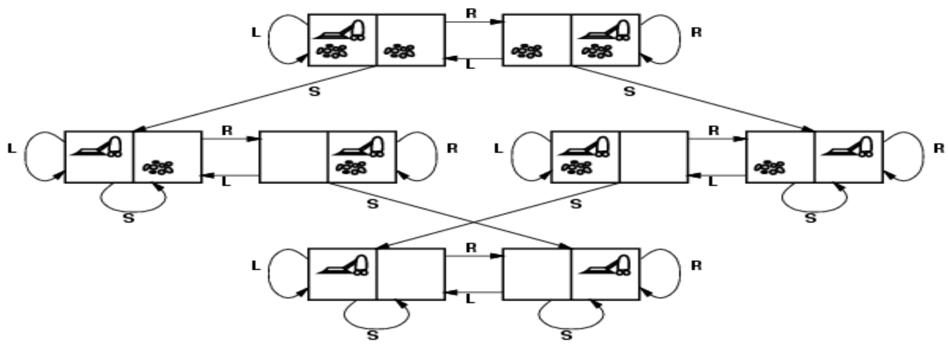
43

Selecting a state space

- (Abstract) solution =
  - set of real paths that are solutions in the real world
- Each abstract action should be "easier" than the original problem

44

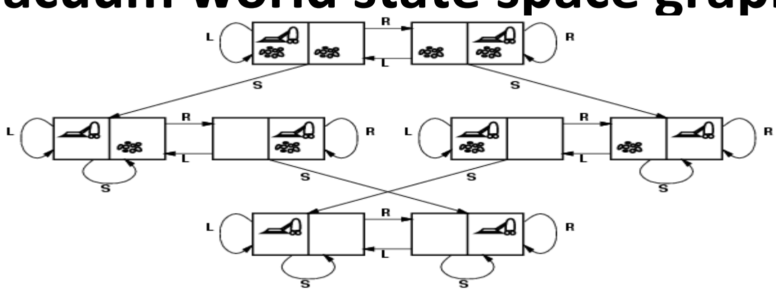
Vacuum world state space graph



- states?
- actions?
- goal test?
- path cost?

45

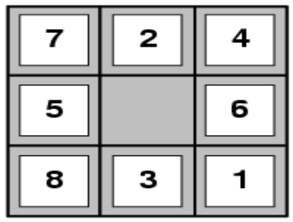
Vacuum world state space graph



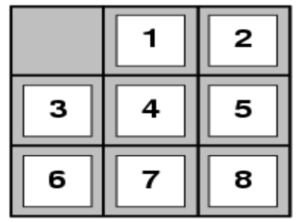
- States: integer dirt and robot location
- Actions? *Left*, *Right*, *Suck*
- Goal test: = no dirt on any location
- Path cost: 1 per action

46

Example: The 8-puzzle



Start State

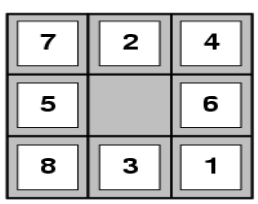


Goal State

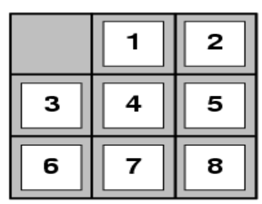
- States?
- Actions?
- Goal test?
- Path cost?

47

Example: The 8-puzzle



Start State

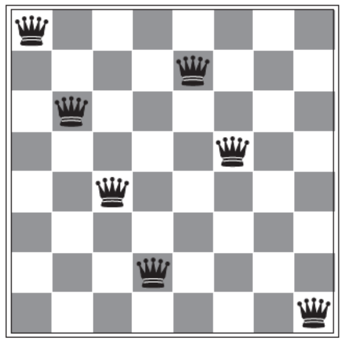


Goal State

- States: locations of tiles
- Actions? move blank left, right, up, down
- Goal test: = goal state (given)
- Path cost: 1 per move

48

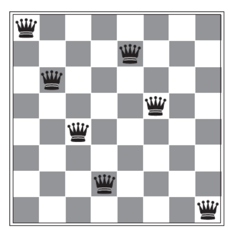
Example: 8 Queens



- States?
- Actions?
- Goal test?
- Path cost?

49

Example: 8 Queens



- States: Any arrangements of 0 to 8 queens on the board
- Actions: Add a queen to any empty square
- Goal test: = 8 queens on the board, none attacked
- Path cost: N/A (we only care about the final state)

50

Real World Problems

- Route finding problem
- Touring problems
  - Visit every city only once, start and end at Bucharest
- Robot navigation
  - Two-dimensional
  - With arms, legs, wheels ➔ many dimensions
- Automatic assembly sequencing

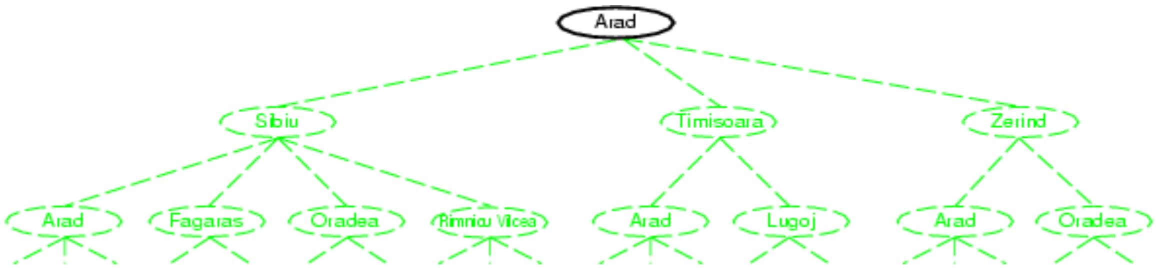
51

Tree search algorithms

- Use a tree analogy for the movement from the initial state to the goal.
- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)

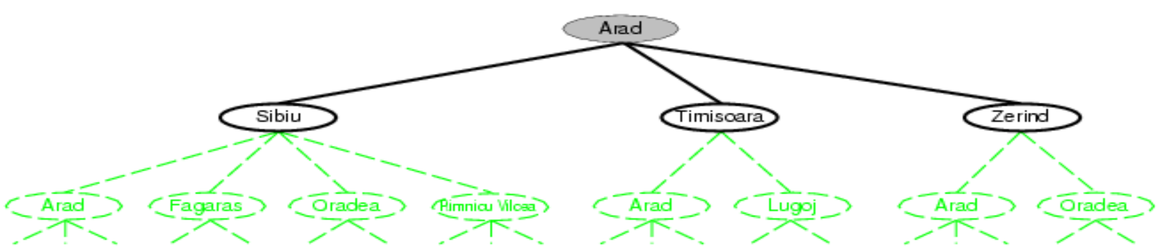
52

Tree search example



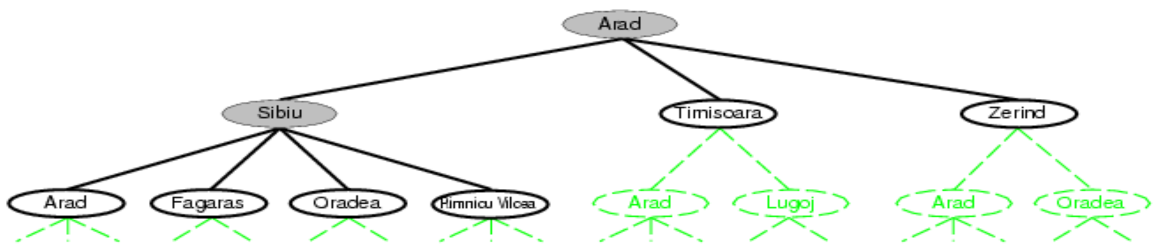
53

Tree search example



54

Tree search example



55

Tree and graph search algorithms

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

56

Tree and graph search algorithms

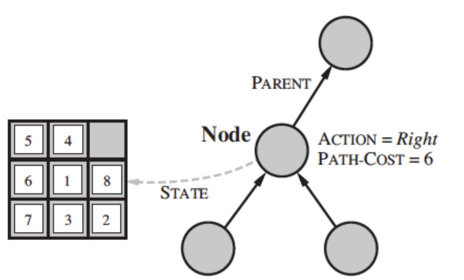
function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

57

Implementation: states vs. nodes

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



58

Search strategies

- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - Completeness:
  - time complexity:
  - space complexity:
  - Optimality:

59

Evaluating a search strategy

- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

60

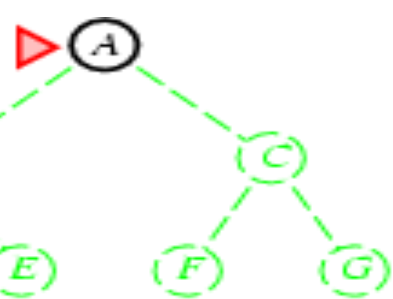
Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search

61

Breadth-first search

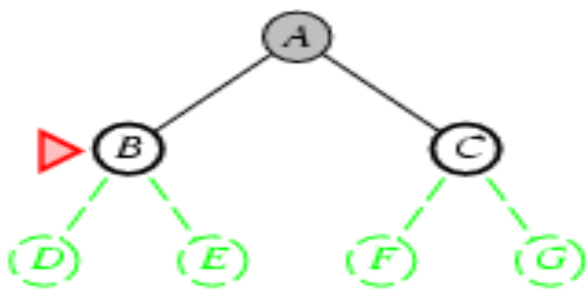
- Expand shallowest unexpanded node
- Implementation:
  - *Frontier* is a FIFO queue, i.e., new successors go at end



62

Breadth-first search

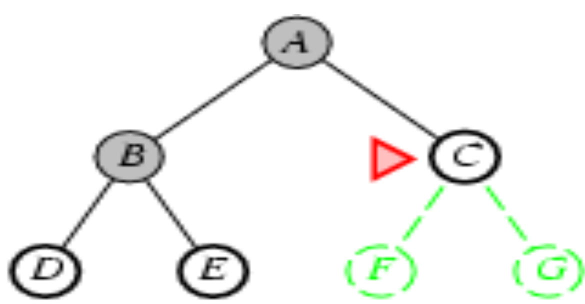
- Expand shallowest unexpanded node
- Implementation:
  - *Frontier* is a FIFO queue, i.e., new successors go at end



63

Breadth-first search

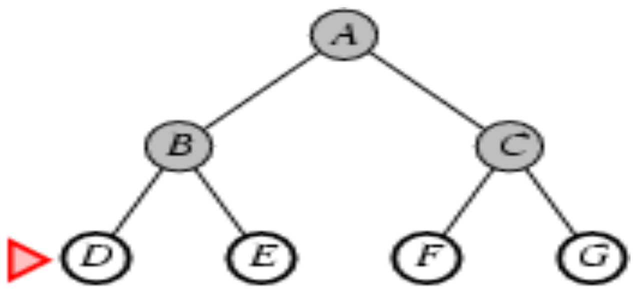
- Expand shallowest unexpanded node
- Implementation:
  - *Frontier* is a FIFO queue, i.e., new successors go at end



64

Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *Frontier* is a FIFO queue, i.e., new successors go at end



65

Properties of breadth-first search

- Complete?
- Time?
- Space?
- Optimal?

66

Properties of breadth-first search

- Complete? Yes (if *b* is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d + b(b^d-1) = O(b^{d+1})$
- Space? $O(b^{d+1})$ (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)

- Space is the bigger problem (more than time)

67

Next time: More search strategies

- Breadth-first search
- **Uniform-cost search**
- **Depth-first search**
- **Depth-limited search**
- **Iterative deepening search**

68