Search

Outline

- **Introduction**
- Problem-solving agents
- Problem formulation
- Problem types
- Example problems
- Basic search algorithms
- Conclusions

Introduction

- Early AI works were directed to:
  - Proof of theorems
  - Solving crosswords
  - Games

- All in AI is search
  - Not entirely true (obviously) but more than you can imagine
  - Finding a good/best solution to a problem among several possible solutions

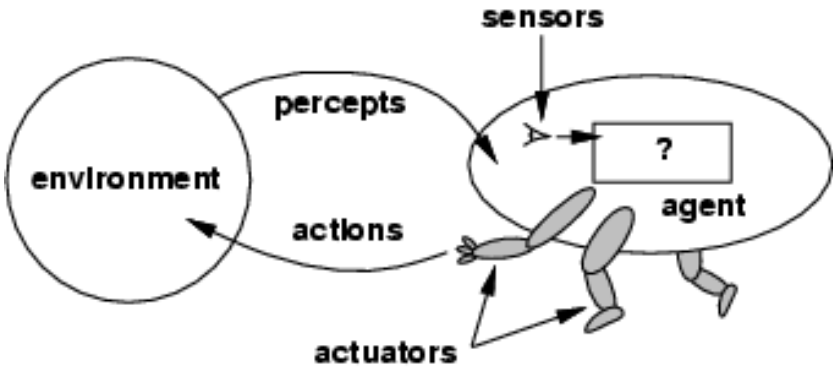
Outline

- Introduction
- **Agents**
- Problem-solving agents
- Problem formulation
- Problem types
- Example problems
- Basic search algorithms
- Conclusions

Agents

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators
- Human agent: eyes, ears, and other organs for sensors; hands, legs, mouth, and other body parts for actuators
- Robotic agent: cameras and infrared range finders for sensors; various motors for actuators

Agents and environments



- The agent function maps from percept histories to actions

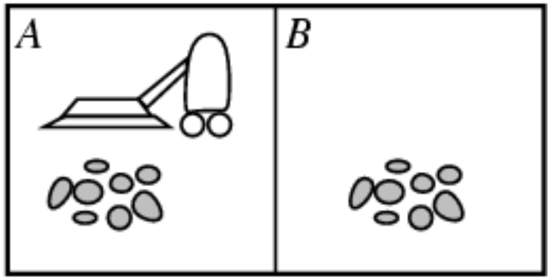
$$[f: \mathcal{P}^{\star} \rightarrow \mathcal{A}]$$

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

Agents

- An agent's behavior depends only on its percept sequence (table of action)
  - Mapping from percept sequences to actions
  - Explicit introduce the list
  - Not exhaustively enumerating it by e.g. a function
- If agent's actions are based completely on built-in knowledge (no attention percepts) → lacks autonomy
- Build behaviors based on own experiences and built-in knowledge

Agents: Vacuum-cleaner world



- Percepts: location and contents, e.g., [A, Dirty]

- Actions: *Left*, *Right*, *Suck*, *NoOp*

Vacuum-cleaner agent

Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
$\vdots$	$\vdots$

function REFLEX-VACUUM-AGENT($[location, status]$) **returns** an action

  if $status = Dirty$ **then return** $Suck$
  else if $location = A$ **then return** $Right$
  else if $location = B$ **then return** $Left$

Outline

- Introduction
- Agents
- **Problem-solving agents**
- Problem formulation
- Problem types
- Example problems
- Basic search algorithms
- Conclusions

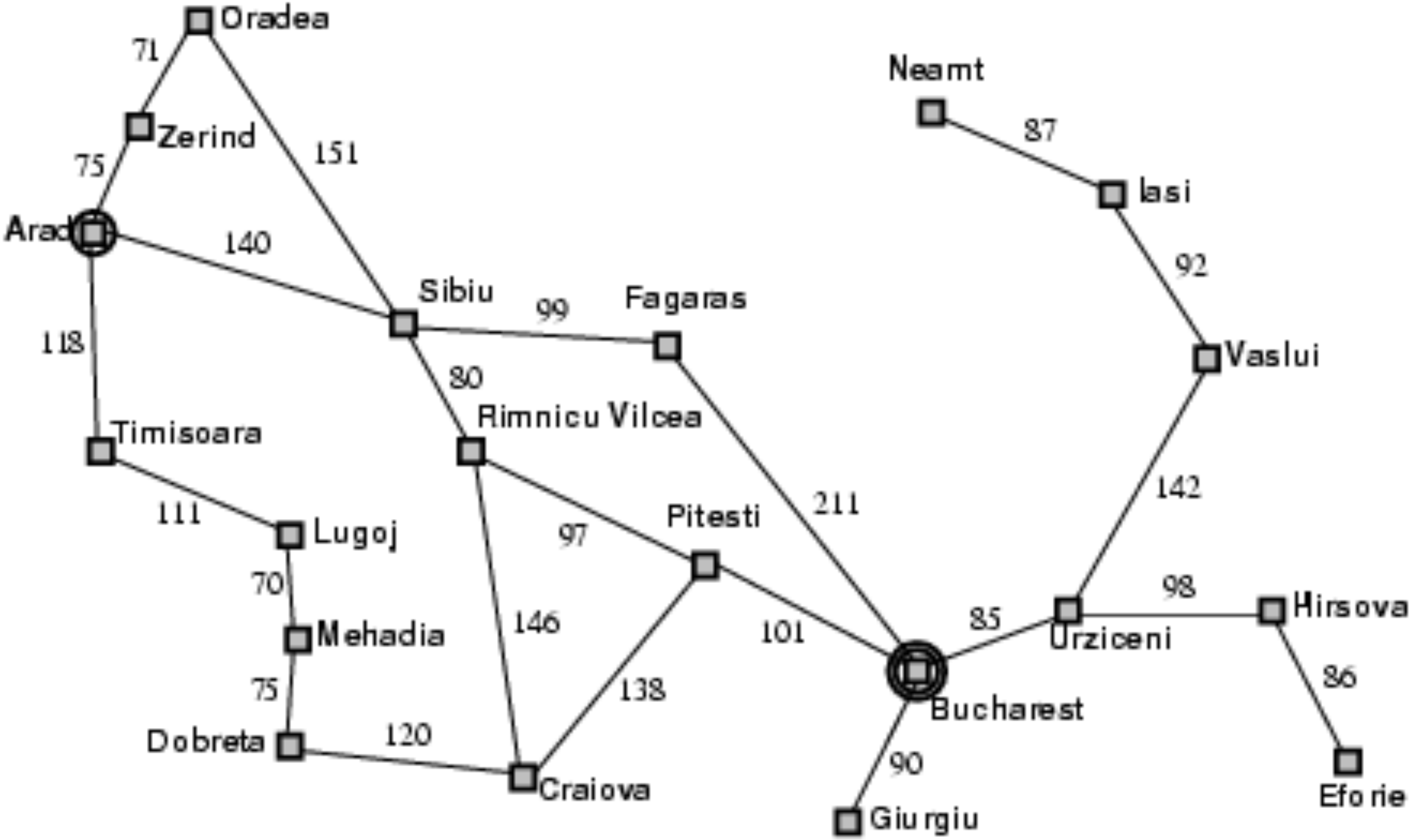
Problem-solving agents (I)

- Agents must maximize its performance measure
- Example: On holiday in Romania; currently in Arad
  Flight leaves tomorrow from Bucharest
- Formulate goal:
  - be in Bucharest
- Formulating the problem:
  - states: multiple cities
  - actions: drive between cities
- Finding a solution:
  - Sequence cities, eg., Arad, Sibiu, Fagaras, Bucharest
- Process of finding such a solution: Search

Problem-solving agents (II)

- Assumptions of the environment:
  - Static: search and formulation is done without considering changes in the environment
  - Observable: the initial state is known
  - Discrete: the alternative locations are known
  - Deterministic: each state is determined by the current state and the action executed
- The solutions are simple sequences of actions, they are executed without considering perceptions

Problem-solving agents (III)



Outline

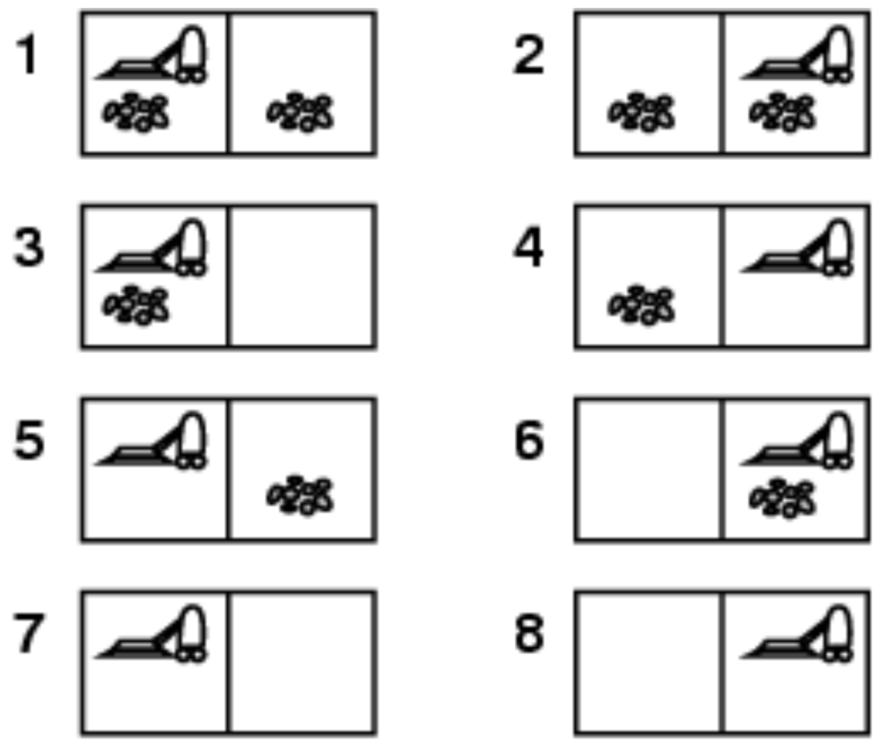
- Introduction
- Agents
- Problem-solving agents
- **Problem types**
- Problem formulation
- Example problems
- Basic search algorithms
- Conclusions

Problem types

- Deterministic, fully observable → single-state problem
  - Agent knows exactly which state it will be in; solution is a sequence
- Non-observable → sensorless problem (conformant problem)
  - Agent may have no idea where it is; solution is a sequence
- Nondeterministic and/or partially observable → contingency problem
  - percepts provide new information about current state
  - often interleave search with execution
- Unknown state space → exploration problem

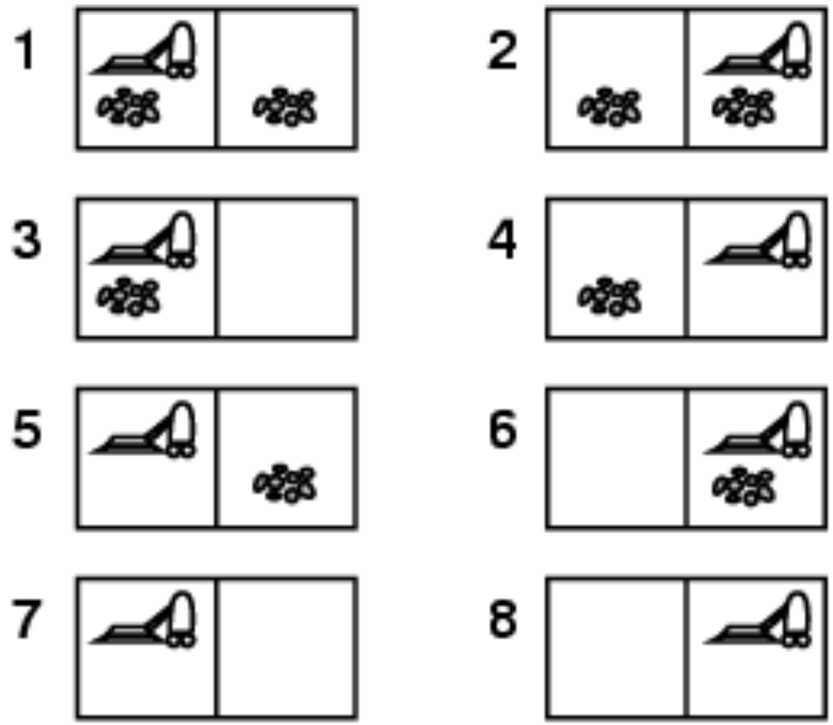
Example: vacuum world

- **Single-state**, start in #5.
  Solution?



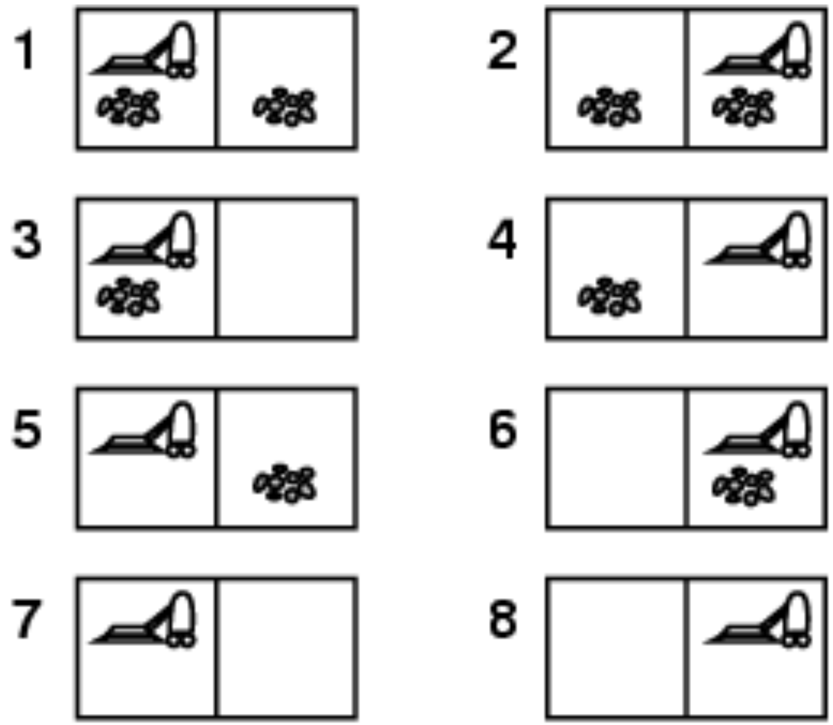
Example: vacuum world

- **Single-state**, start in #5.
  Solution? *[Right, Suck]*
- **Sensorless**, start in {*1,2,3,4,5,6,7,8*} e.g., *Right* goes to {*2,4,6,8*}
  Solution?



Example: vacuum world

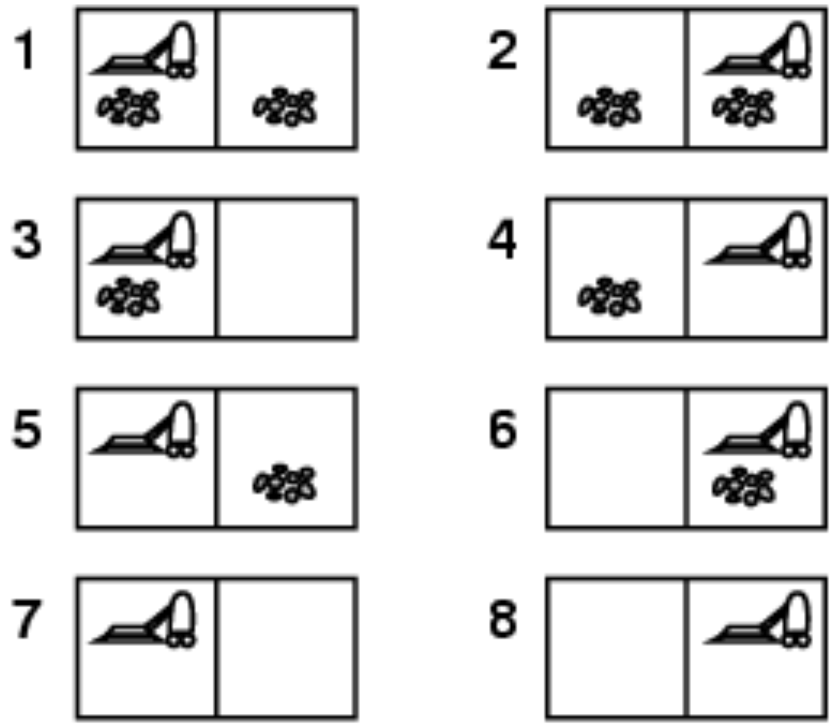
- **Sensorless,** start in {*1,2,3,4,5,6,7,8*} e.g., *Right* goes to {*2,4,6,8*} Solution? *[Right,Suck,Left,Suck]*



- **Contingency**
  - Nondeterministic: *Suck* may dirty a clean carpet
  - Partially observable: location, dirt at current cell location.
  - Percept: *[L, Clean],* i.e., start in #5 or #7 Solution?

Example: vacuum world

- **Sensorless**, start in {*1,2,3,4,5,6,7,8*} e.g., *Right* goes to {*2,4,6,8*}
  Solution?
  [Right,Suck,Left,Suck]



- **Contingency**
  - Nondeterministic: *Suck* may dirty a clean carpet
  - Partially observable: location, dirt at current location.
  - Percept: *[L, Clean],* i.e., start in #5 or #7
    Solution? *[Right, **if** dirt **then** Suck]*

Outline

- Introduction
- Agents
- Problem-solving agents
- Problem types
- **Problem formulation**
- Example problems
- Basic search algorithms
- Conclusions

Problem formulation

- A problem is defined by four items:

1. initial state e.g., "at Arad"
2. actions or successor function $S(x)$ = set of action–state pairs
   - e.g., $S(Arad) = \{$<*Arad → Zerind, Zerind*>, … $\}$
3. goal test, can be
   - explicit, e.g., x = "at Bucharest"
   - implicit, e.g., *Checkmate(x)*
4. path cost (additive)
   - e.g., sum of distances, number of actions executed, etc.
   - $c(x,a,y)$ is the step cost, assumed to be ≥ 0

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

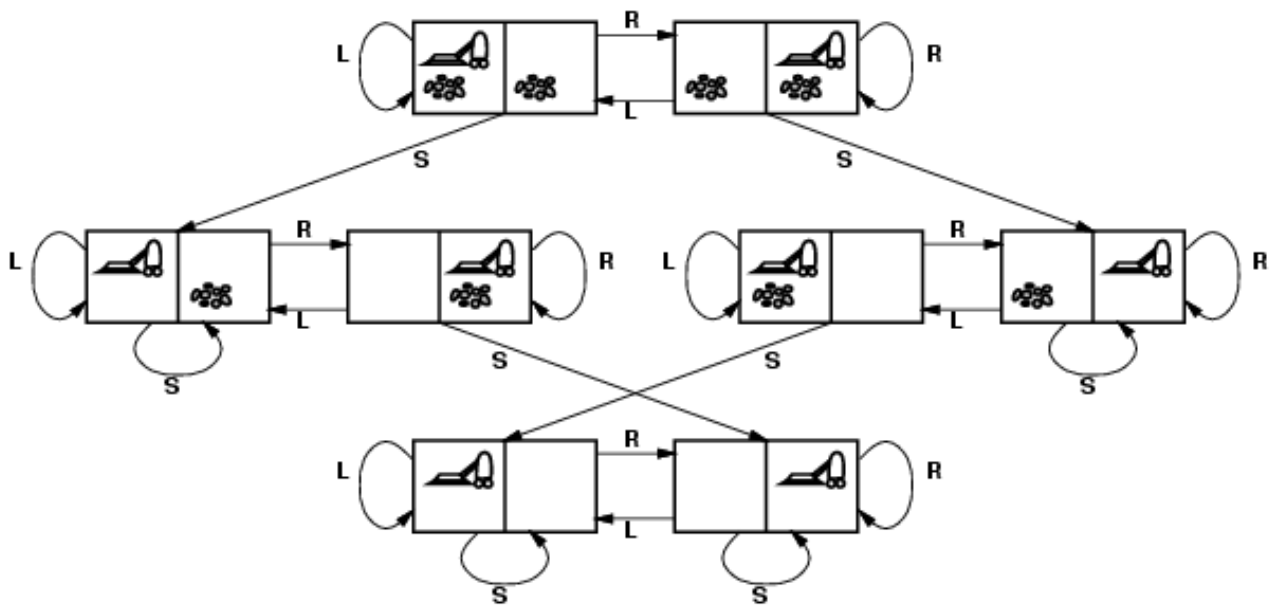
Problem formulation

- Real world is absurdly complex
  → state space must be abstracted for problem solving
- (Abstract) state = set of real states
- (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”
- (Abstract) solution =
  - set of real paths that are solutions in the real world
- Each abstract action should be "easier" than the original problem

Outline

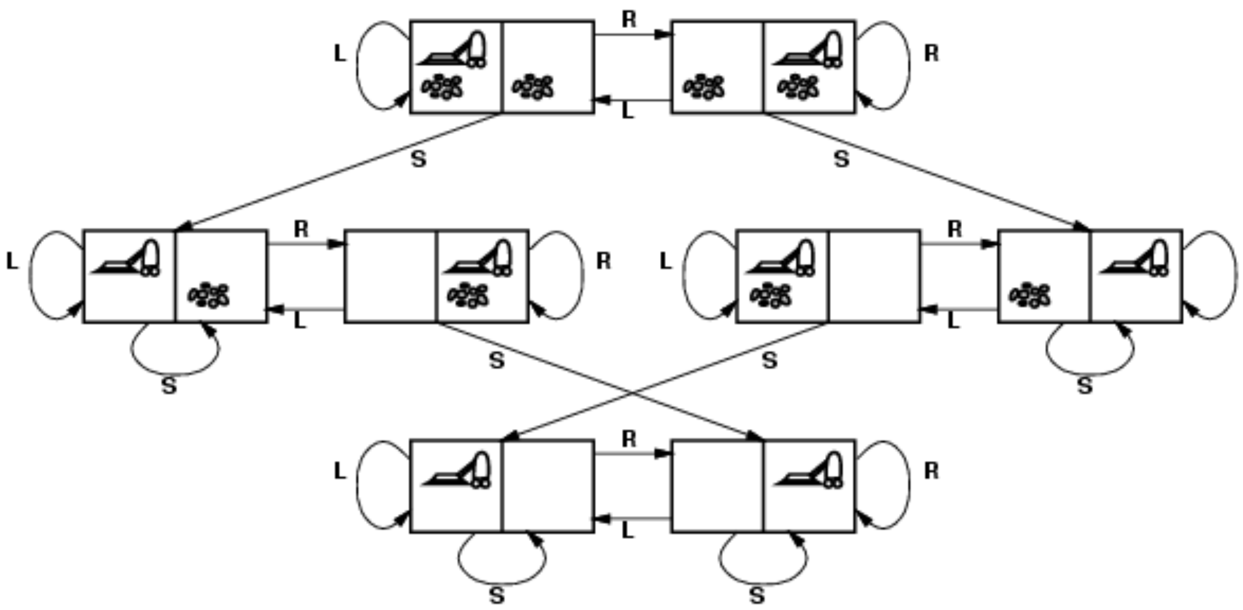
- Introduction
- Agents
- Problem-solving agents
- Problem types
- Problem formulation
- **Example problems**
- Basic search algorithms
- Conclusions

Vacuum world state space graph



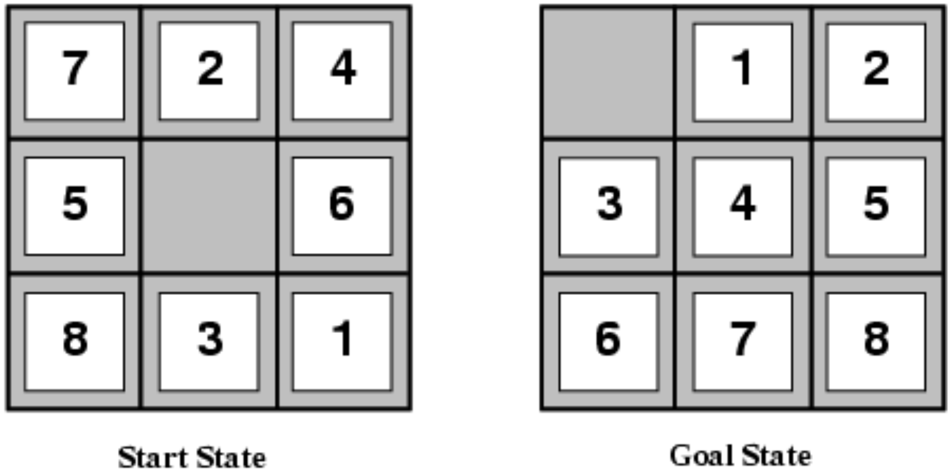
- states?
- actions?
- goal test?
- path cost?

Vacuum world state space graph



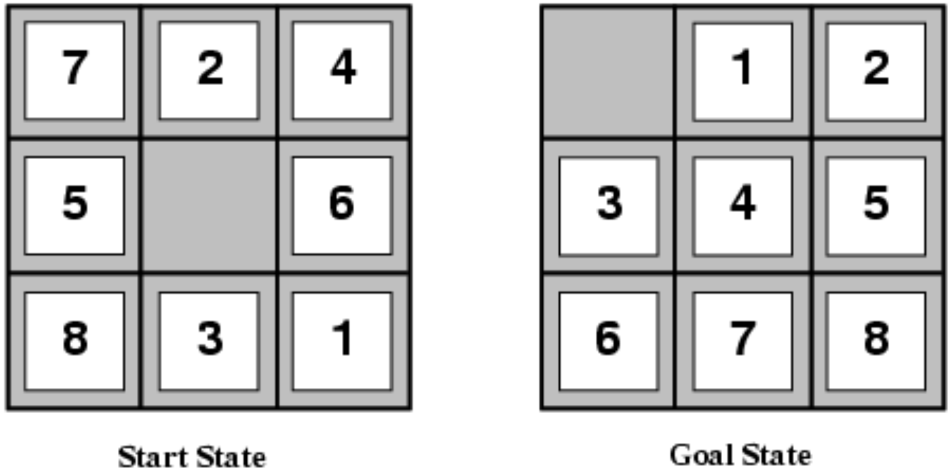
- states? integer dirt and robot location
- actions? *Left*, *Right*, *Suck*
- goal test? no dirt at all locations
- path cost? 1 per action

Example: The 8-puzzle



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

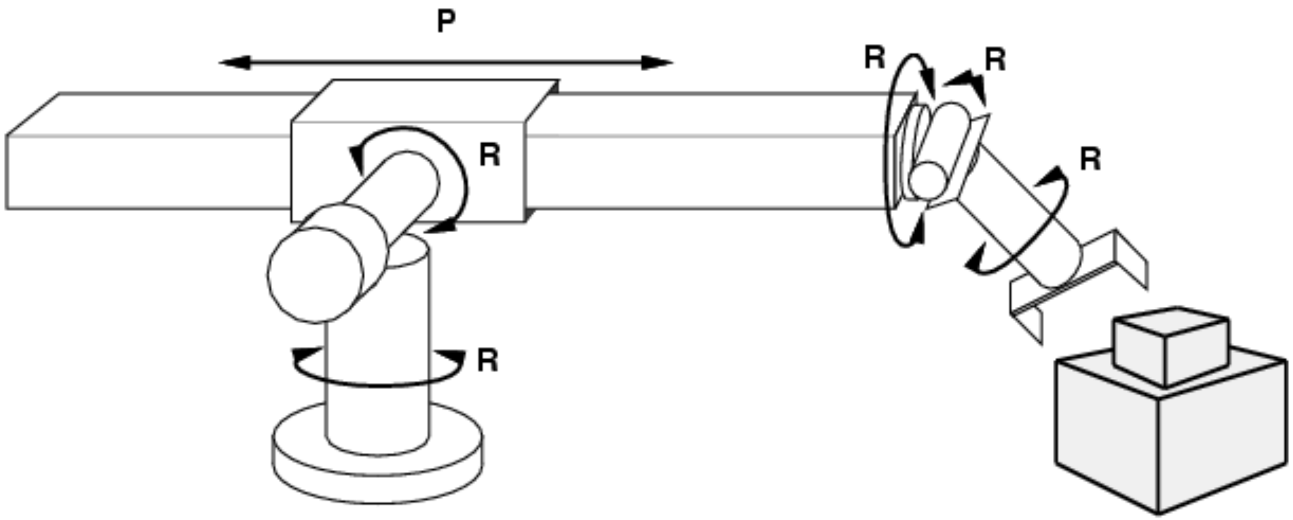
Example: The 8-puzzle



- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move

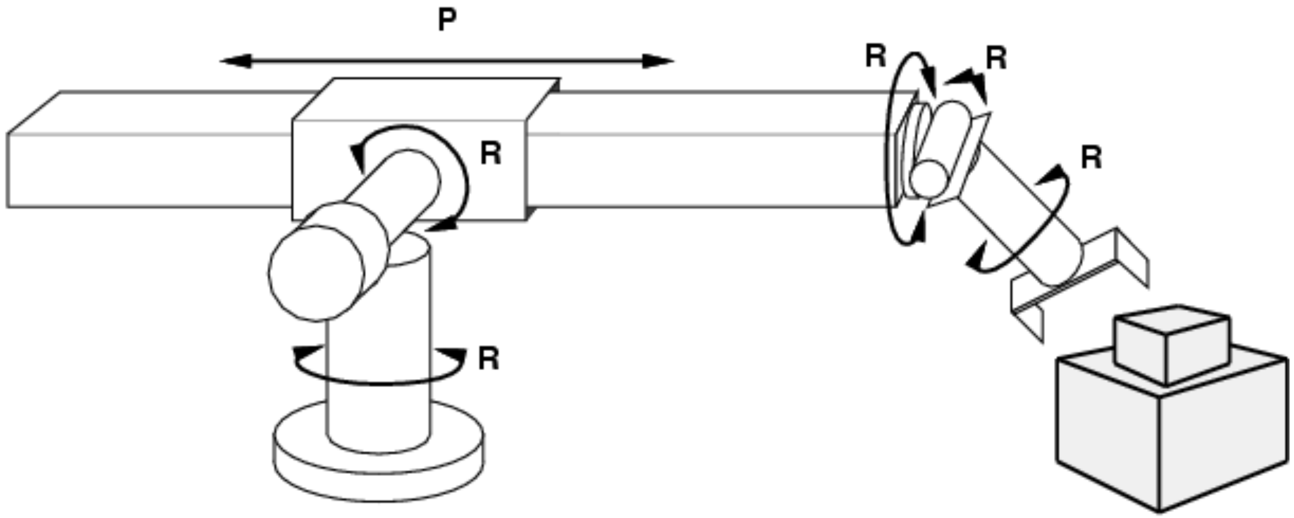
[Note: optimal solution of *n*-Puzzle family is NP-hard]

Example: robotic assembly



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

Example: robotic assembly



- states?: real-valued coordinates of robot joint angles parts of the object to be assembled
- actions?: continuous motions of robot joints
- goal test?: complete assembly
- path cost?: time to execute

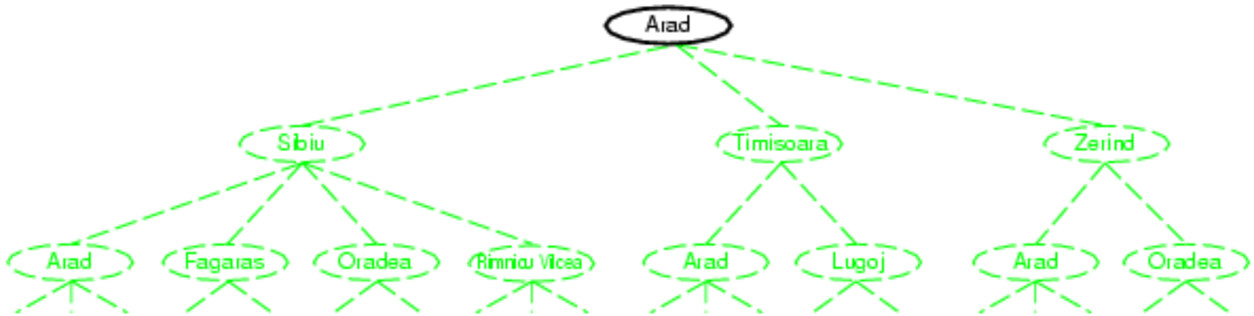
Outline

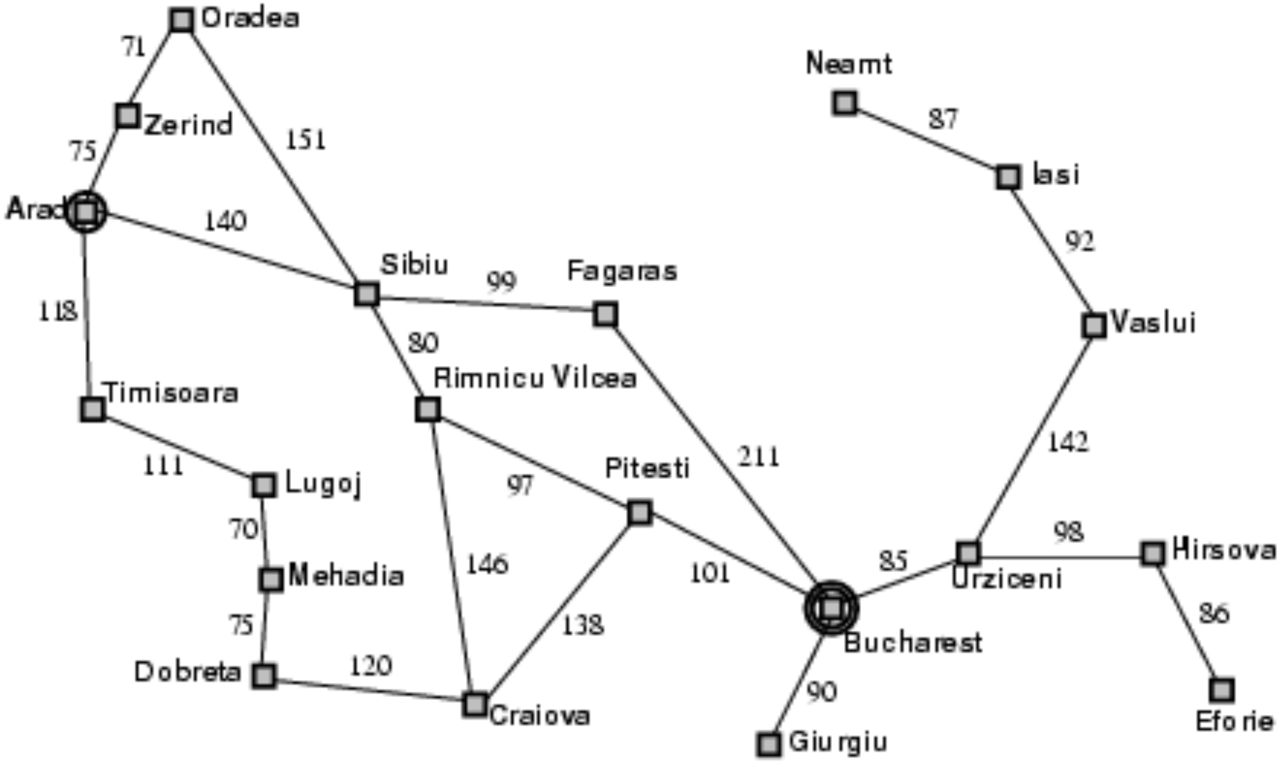
- Introduction
- Agents
- Problem-solving agents
- Problem types
- Problem formulation
- Example problems
- **Basic search algorithms**
- Conclusions

Search algorithms

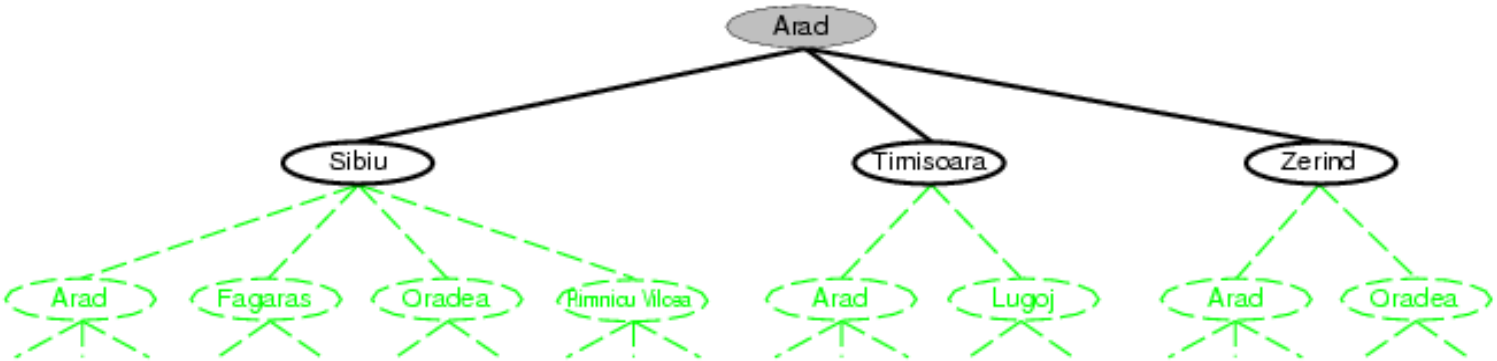
- We have formulated problems, we now need to solve them: search tree
- In general we can have a search graph rather than a tree when the state can be reached from multiple paths
- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)

Tree search example

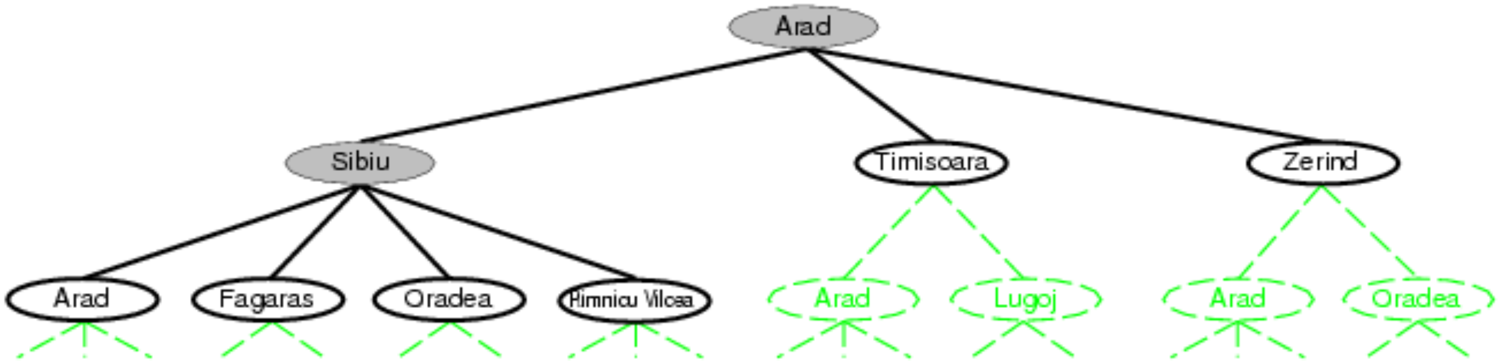




Tree search example

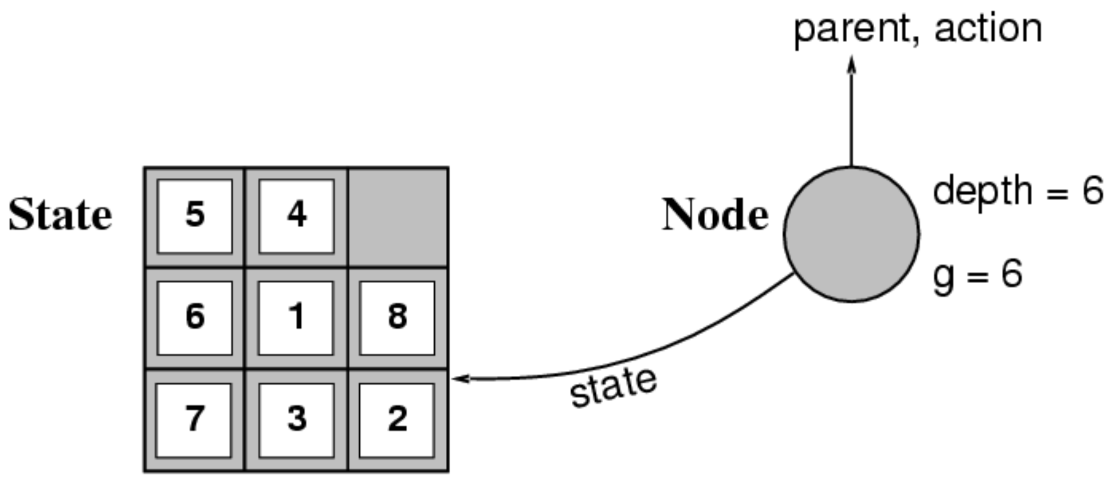


Tree search example



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



- The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.

Search strategies

- A search 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
  - 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 ∞)

Outline

- Introduction
- Agents
- Problem-solving agents
- Problem types
- Problem formulation
- Example problems
- Basic search algorithms
  - **Uninformed search strategies**
  - Informed search strategies
- Conclusions

Outline

- Introduction
- Agents
- Problem-solving agents
- Problem types
- Problem formulation
- Example problems
- Basic search algorithms
  - Uninformed search strategies
  - **Informed search strategies**
- Conclusions

Outline

- Introduction
- Agents
- Problem-solving agents
- Problem formulation
- Problem types
- Example problems
- Basic search algorithms
- **Conclusions**

Conclusions

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored

- Variety of uninformed and informed search strategies