Solving Problems by Search:
Problem Formulation and Uninformed Search

CZ3005: Artificial Intelligence

Shen Zhiqi

NANYANG TECHNOLOGICAL UNIVERSITY

Outline

- How to define a problem-solving agent?
- Types of problem
- Problem formulation
- Informed search strategies

Problem-Solving Agent

Rational goal-based agent:

- Performance measure defined in terms of satisfying goals

Goal formulation

- Define and organize objectives (goal states)

Problem formulation

- Define what states and actions (transitions) to consider

Search for a solution

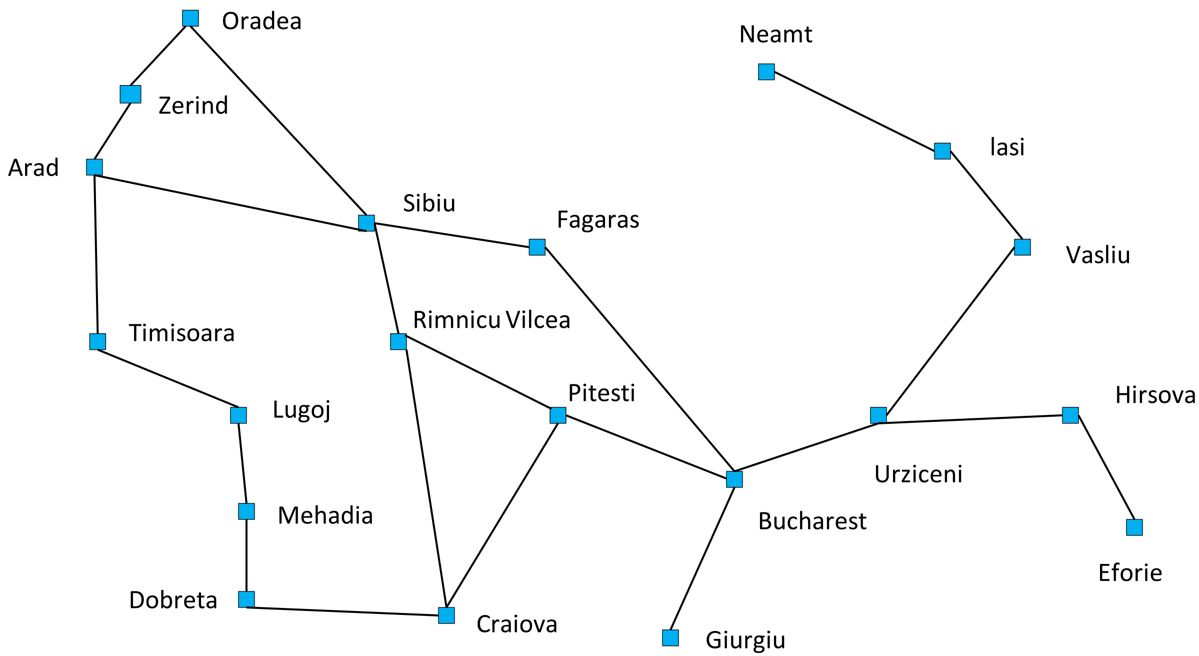
- Find a sequence of actions that lead to a goal state

Execution

- Actually carry out the recommended actions

Goal-based Agent: Example

On holiday in Romania



- Currently in Arad (Initial state). Flight leaves tomorrow from Bucharest.
- Goal: be in Bucharest (other factors: cost, time, most scenic route, etc)
- State: be in a city (defined by the map)
- Action: transition between states (highways defined by the map)

Design of Problem-Solving Agent

Idea:

- Systematically considers the expected outcomes of different possible sequences of actions that lead to states of known value
- Choose the best one
  - shortest journey from A to B?
  - most cost effective journey from A to B?

Steps:

1. Goal formulation
2. Problem formulation
3. Search process

   No knowledge → uninformed search

   Knowledge → informed search
4. Action execution (follow the recommended route)

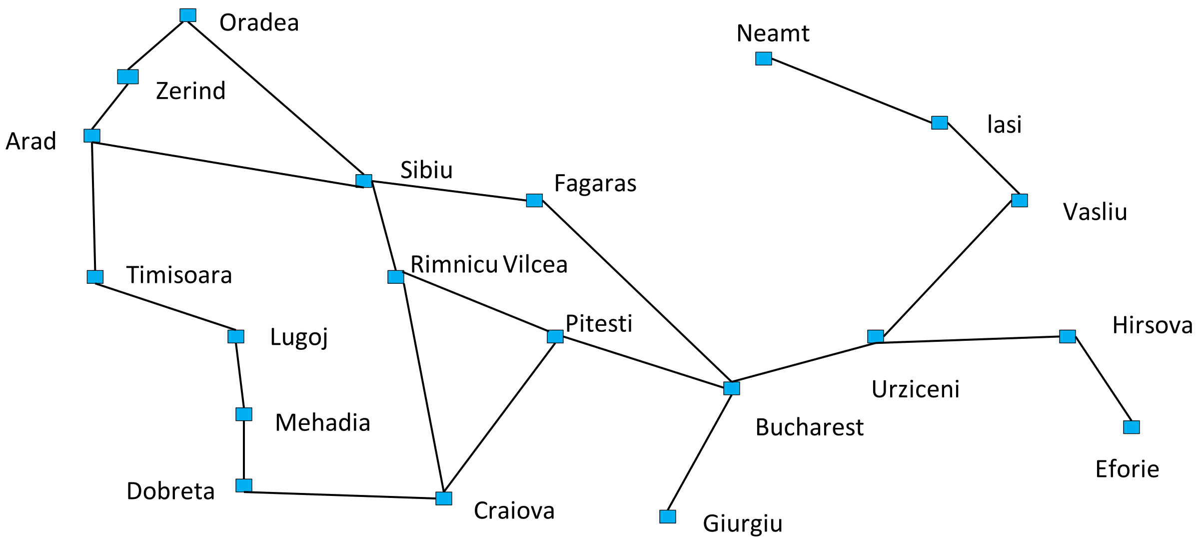
Algorithm

```
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
```

Example: Romania

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

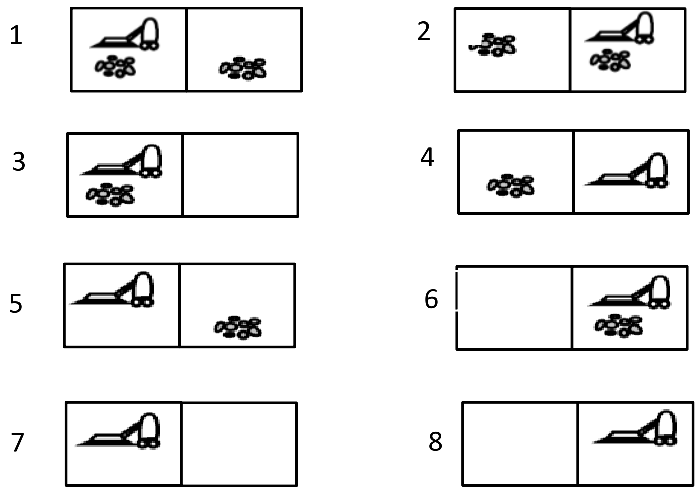


Example: Vacuum Cleaner Agent

- Robotic vacuum cleaners move autonomously
- Some can come back to a docking station to charge their batteries
- A few are able to empty their dust containers into the dock as well

Example: A Simple Vacuum World

- Two locations, each location may or may not contain dirt, and the agent may be in one location or the other



- 8 possible world states
- Possible actions: left, right, and suck
- Goal: clean up all dirt → Two goal states, i.e. {7, 8}

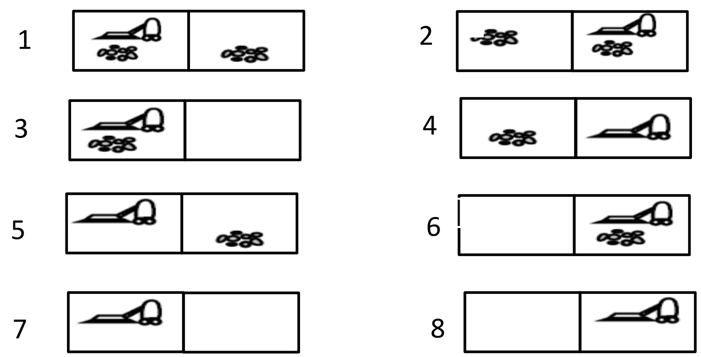
Problem Types

The problem type depends on how much knowledge the agent has concerning its actions and the state that it is in.

- Single-State Problem
- Multiple-State Problem
- Contingency Problem

Single-State Problem

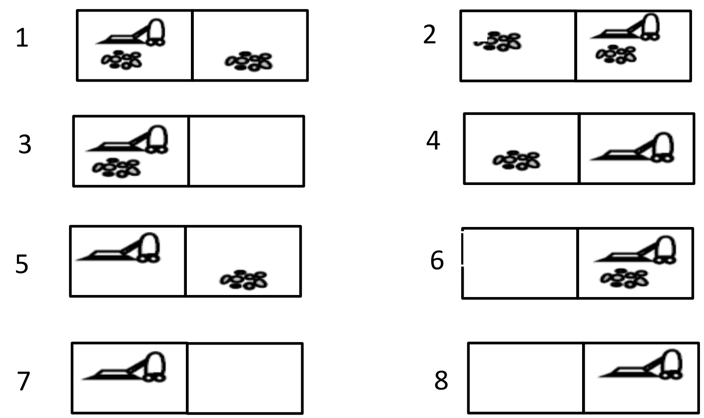
- Accessible world state (sensory information is available)
- Known outcome of action (deterministic)



- e.g.: start in #5
  - Solution: **right, suck**

Multiple-State Problem

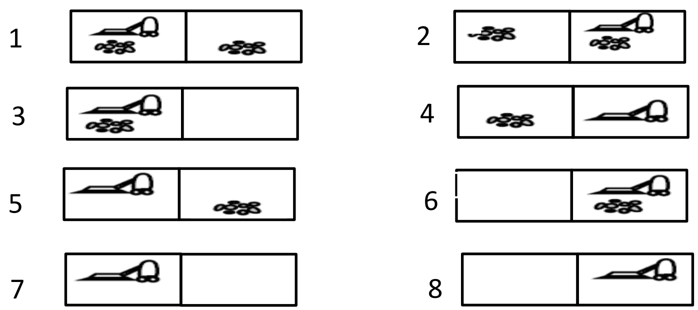
- Inaccessible world state (with limited sensory information): agent only knows which sets of states it is in
- Known outcome of action (deterministic)



- e.g.: start in {1, 2, 3, 4, 5, 6, 7, 8}
  - Action **right** goes to {2, 4, 6, 8}
  - Solution: **right, suck, left, suck**

Contingency Problem

- Limited or no sensory information (inaccessible)
- Limited agent knowledge, action result is not predictable
- Effect of action depends on what is found to be true through perception/monitoring (non-deterministic)
  - Suppose action **suck** may deposits dirt at a clean carpet



  - e.g. start in #4, **suck** → reach {2,4}, solution: **left**, suck
  - e.g. start in {1, 3} and have only local dirt sensor ,solution:
    - **suck, right**
    - **suck** only if there is dirt there
- problem solving requires sensing during the execution phase
  - e.g., keep your eyes open while walking(contingency)

Well-Defined Formulation

Definition of a problem:

- The information used by an agent to decide what to do

Specification:

- Initial state
- Action set, i.e. available actions (successor functions)
- State space, i.e. states reachable from the initial state
  - Solution path: sequence of actions from one state to another
- Goal test predicate
  - Single state, enumerated list of states, abstract properties
- Cost function
  - Path cost *g(n),* sum of all (action) step costs along the path

Solution:

- A path (a sequence of operators leading) from the Initial-State to a state that satisfies the Goal-Test

Measuring Problem-Solving Performance

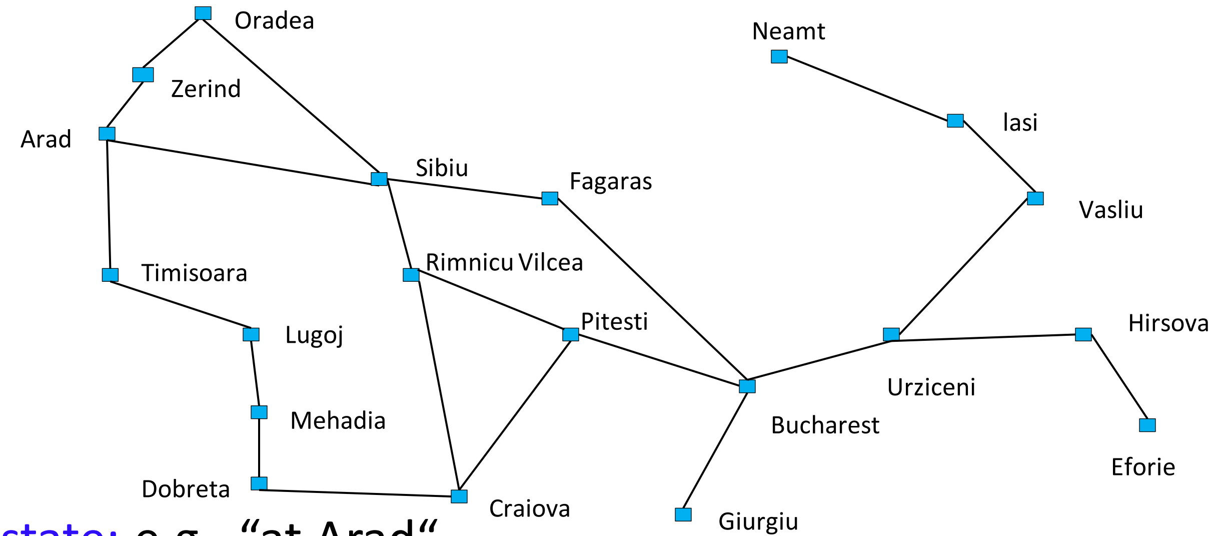
Search Cost:

- What does it cost to find the solution?
  - e.g. How long (time)? How many resources used (memory)?

Total cost of problem-solving

- Search cost (“offline") + Execution cost (“online")
- Trade-offs often required
  - Search a very long time for the optimal solution, or
  - Search a shorter time for a “good enough" solution

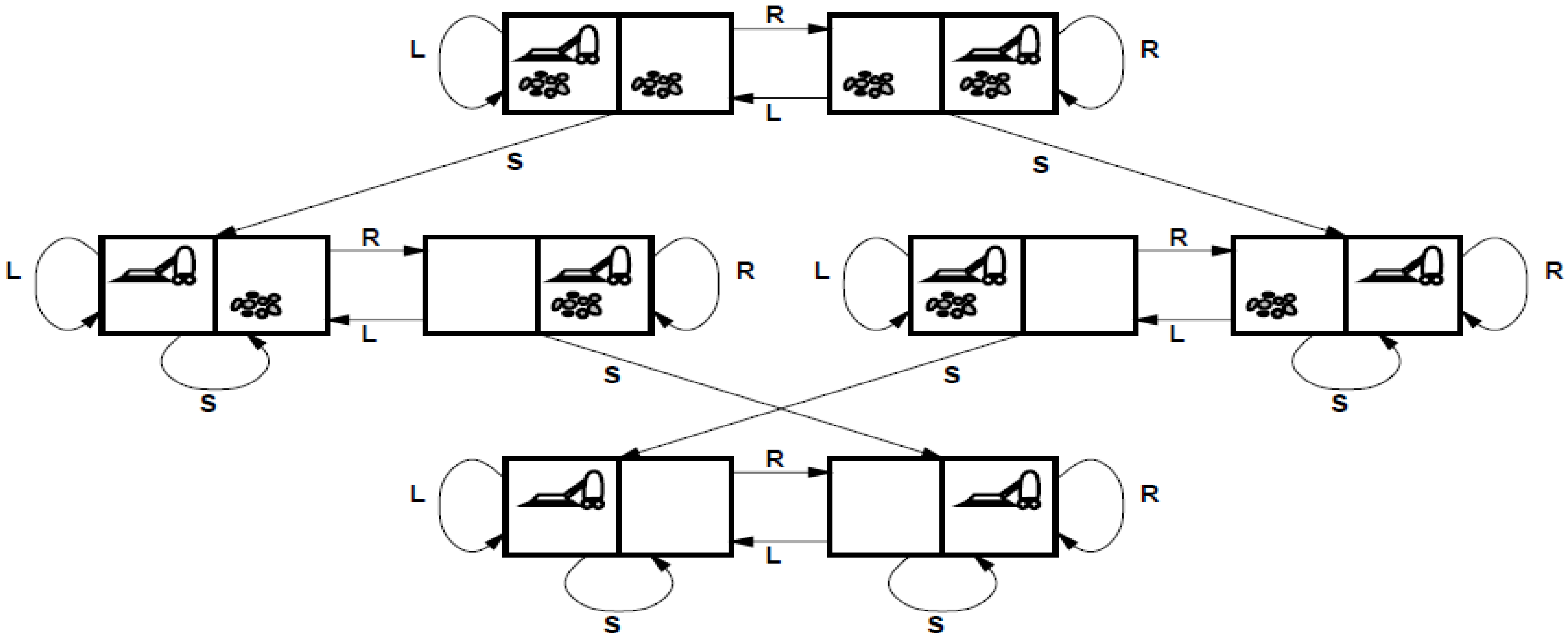
Single-State Problem Example



- Initial state: e.g., “at Arad“
- Set of possible actions and the corresponding next states
  - e.g., Arad → Zerind
- Goal test:
  - explicit (e.g., x = “at Bucharest")
- Path cost function
  - e.g., sum of distances, number of operators executed solution: a sequence of operators leading from the initial state to a goal state

Example: Vacuum World (Single-state Version)

- Initial state: one of the eight states shown previously
- Actions: left, right, suck
- Goal test: no dirt in any square
- Path cost: 1 per action



Multiple-State Problem Formulation

- Initial state set
- Set of possible actions and the corresponding sets of next states
- Goal test
- Path cost function

Solution:

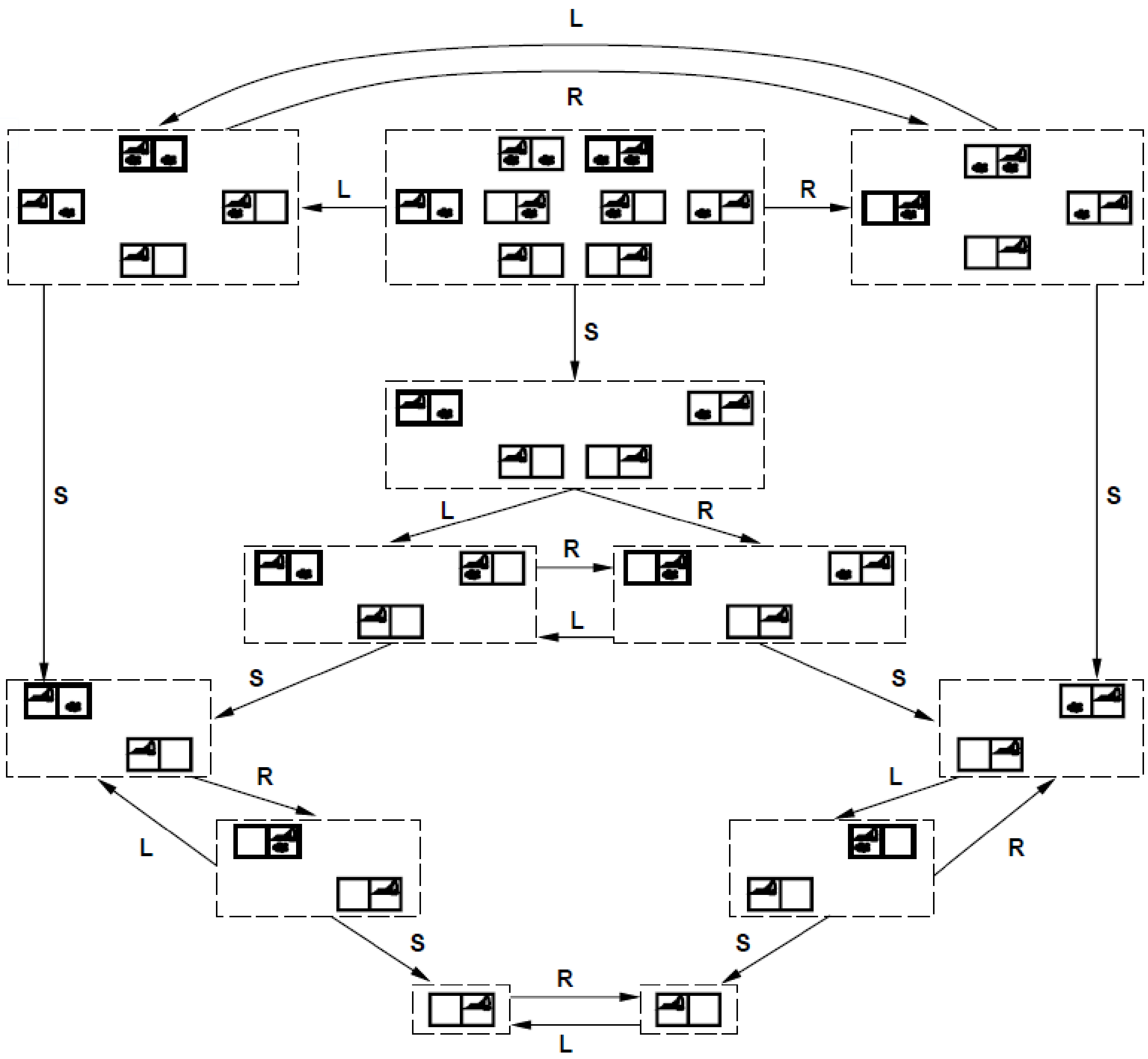
- a path (connecting sets of states) that leads to a set of states

all of which are goal states

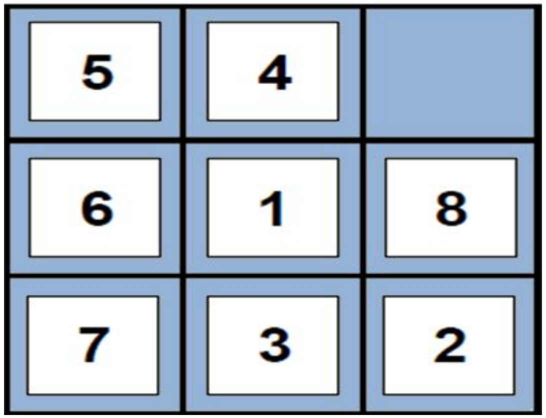
Example: Vacuum World (Multiple-state Version)

- States: subset of the eight states
- Operators: left, right, suck
- Goal test: all states in state set have no dirt
- Path cost: 1 per operator

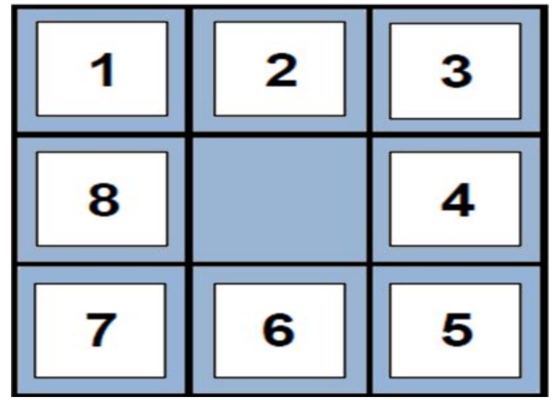
Example: Vacuum World (Multiple-state Version)



Example: 8-puzzle



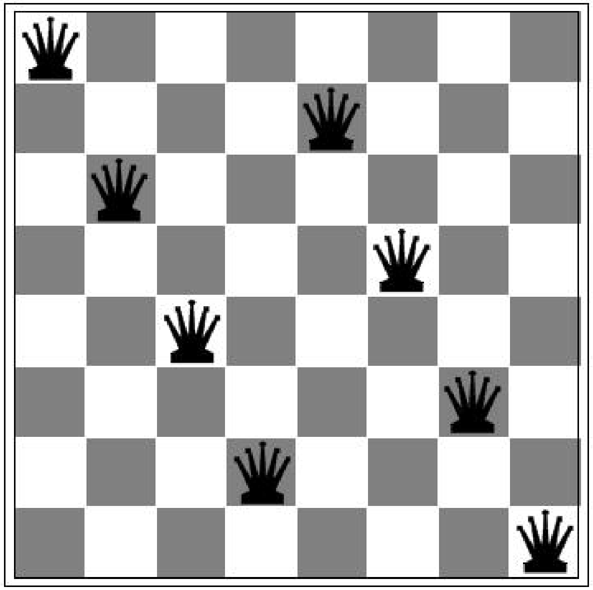
Start state



Goal state

- States: integer locations of tiles
  - number of states = 9!
- Actions: move blank left, right, up, down
- Goal test: = goal state (given)
- Path cost: 1 per move

Example: 8-queens



- States: Any arrangement of 0 to 8 queens on the board
- Actions: Add a queen to any empty square
- Goal test: 8 queens are on the board, none attacked
- Path cost: Not necessary

Real-World Problems

Route finding problems:

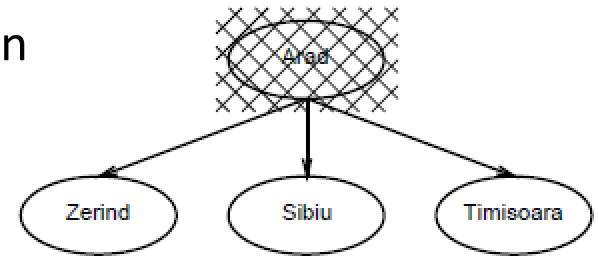
- Routing in computer networks
- Robot navigation
- Automated travel advisory
- Airline travel planning

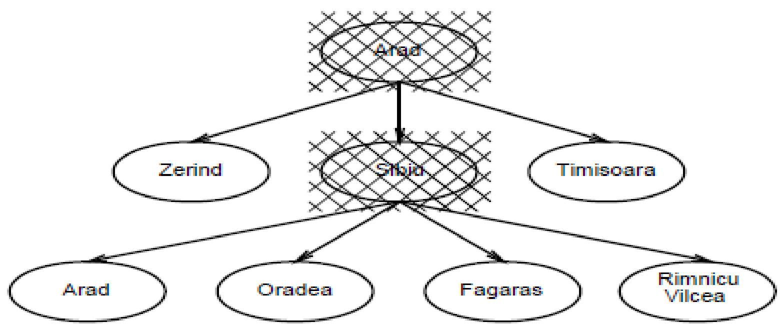
Touring problems:

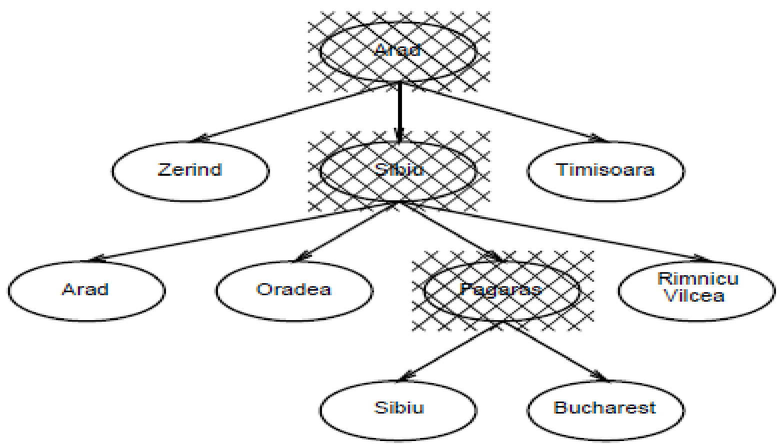
- Traveling Salesperson problem
- “Shortest tour": visit every city exactly once

Search Algorithms

- Exploration of state space by generating successors of already-explored states
- Expanding the states
  - Frontier: candidate nodes for expansion
  - Explored set







Search Algorithms...

```
Function GENERAL-SEARCH(problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
          else expand the node and add the resulting nodes to the search tree
    end
```

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
  - how long does it take to find a solution: the number of nodes generated
- space complexity
  - maximum number of nodes in memory
- optimality
  - does it always find the best (least-cost) solution?

Branching factor

- Maximum number of successors of any node
- Or average branching factor (Tutorial Q1(c))

Uninformed vs Informed

Uninformed search strategies

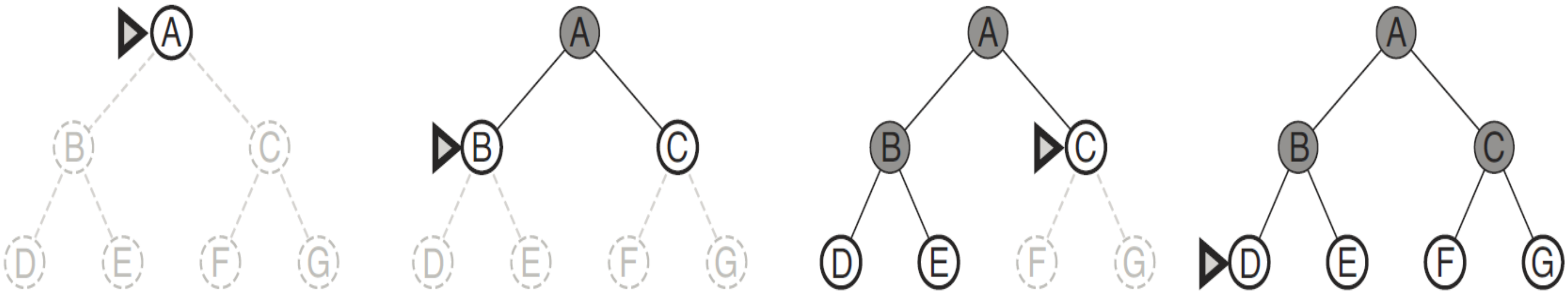
- use only the information available in the problem definition
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Informed search strategies

- use problem-specific knowledge to guide the search
- usually more efficient

Breadth-First Search

Expand shallowest unexpanded node which can be implemented by a First-In-First-Out (FIFO) queue



Denote

- b: maximum branching factor of the search tree
- d: depth of the least-cost solution

Complete: Yes

Optimal: Yes when all step costs equally

Complexity of BFS

Branching factor

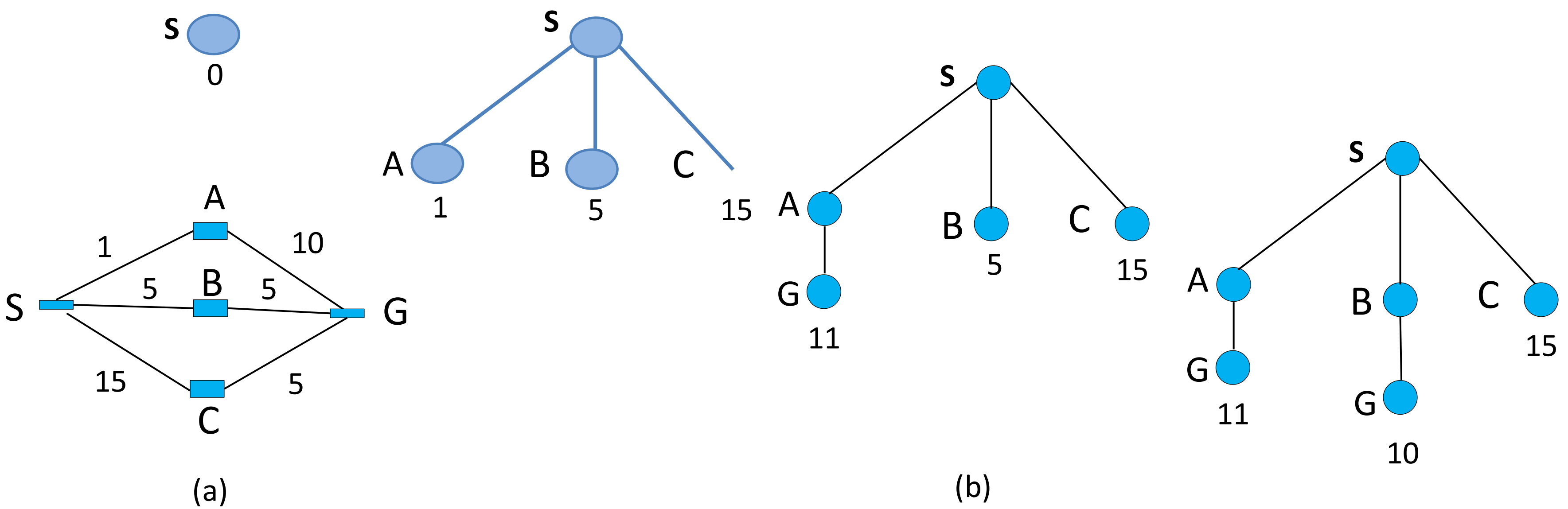
- Hypothetical state-space, where every node can be expanded into *b* new nodes, solution of path-length *d*
- Time: $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$
- Space: (keeps every node in memory) $O(b^d)$ are equal

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11111 terabytes

Uniform-Cost Search

To consider edge costs, expand unexpanded node with the least path cost *g*

- Modification of breath-first search
- Instead of First-In-First-Out (FIFO) queue, using a priority queue with path cost *g(n)* to order the elements
- BFS = UCS with *g(n)* = Depth(n)



Uniform-Cost Search...

- Complete: Yes
- Time: # of nodes with path cost g <= cost of optimal solution (eqv. # of nodes pop out from the priority queue)
- Space: # of nodes with path cost g <= cost of optimal solution
- Optimal: Yes

Depth-First Search

Denote

- m: maximum depth of the state space

Complete:

- infinite-depth spaces: No
- finite-depth spaces with loops: No
  - with repeated-state checking: Yes
- finite-depth spaces without loops: Yes

Time: $O(b^m)$

- if solutions are dense, may be much faster than breadth-first

Space: $O(bm)$

Optimal: No

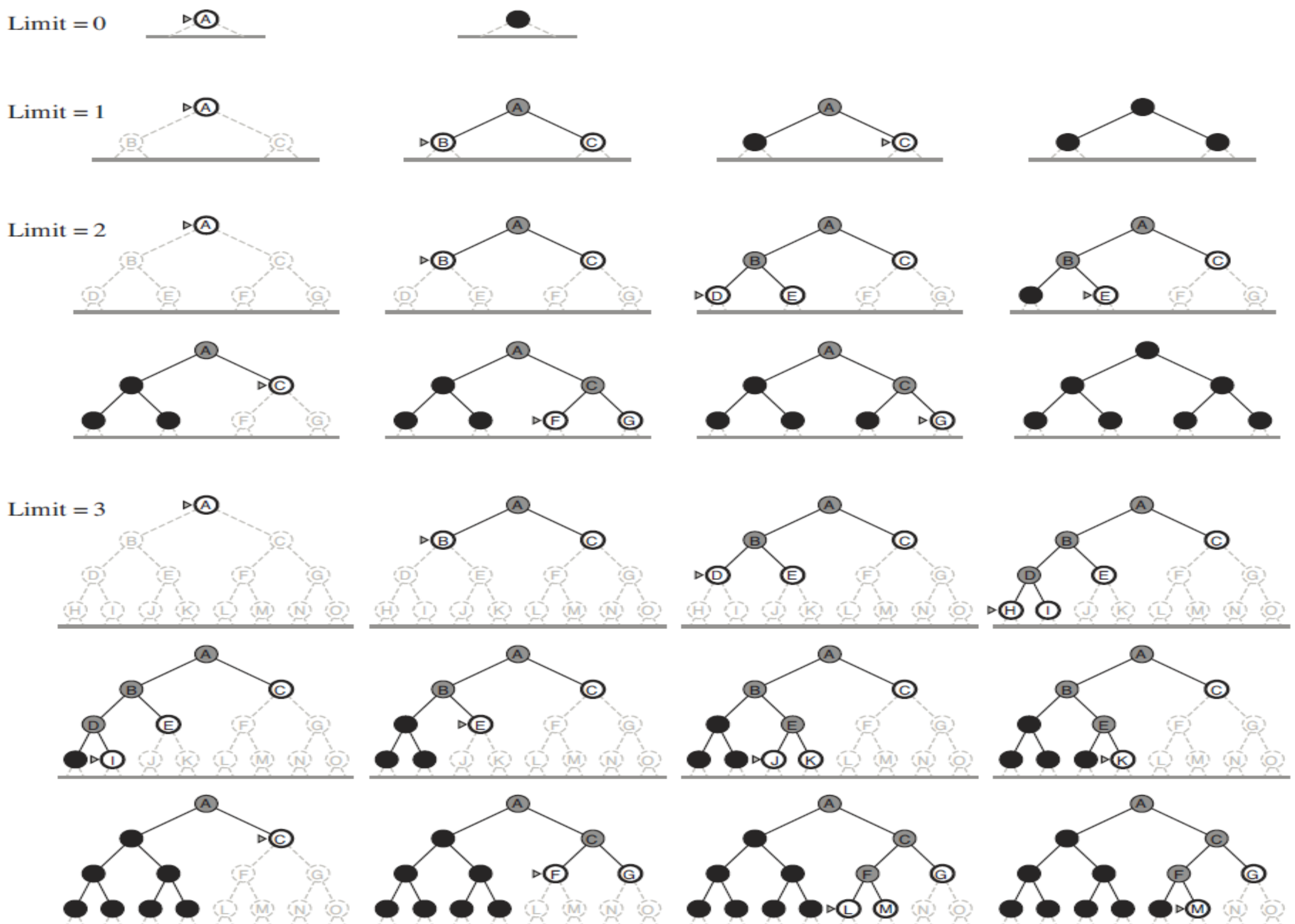
Depth-Limited Search

To avoid infinite searching, Depth-first search with a cutoff on the max depth / of a path

- Complete: Yes, if $I \geq d$
- Time: $O(b^I)$
- Space: $O(bI)$
- Optimal: No

Iterative Deepening Search

Iteratively estimate the max depth l of DLS one-by-one



Iterative Deepening Search...

```
Function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution sequence
   inputs: problem, a problem

   for depth   0 to ∞ do
    if DEPTH-LIMITED-SEARCH(problem, depth) succeeds then return its result
   end
   return failure
```

- Complete: Yes
- Time: $O(b^d)$
- Space: $O(bd)$
- Optimal: Yes

Summary (we make assumptions for optimality)

Criterion	Breadth-first	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional(if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal	Yes	Yes	No	No	Yes	Yes
Complete	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes