Artificial Intelligence

COURSE INSTRUCTOR: MUHAMMAD SAIF UL ISLAM

Outline

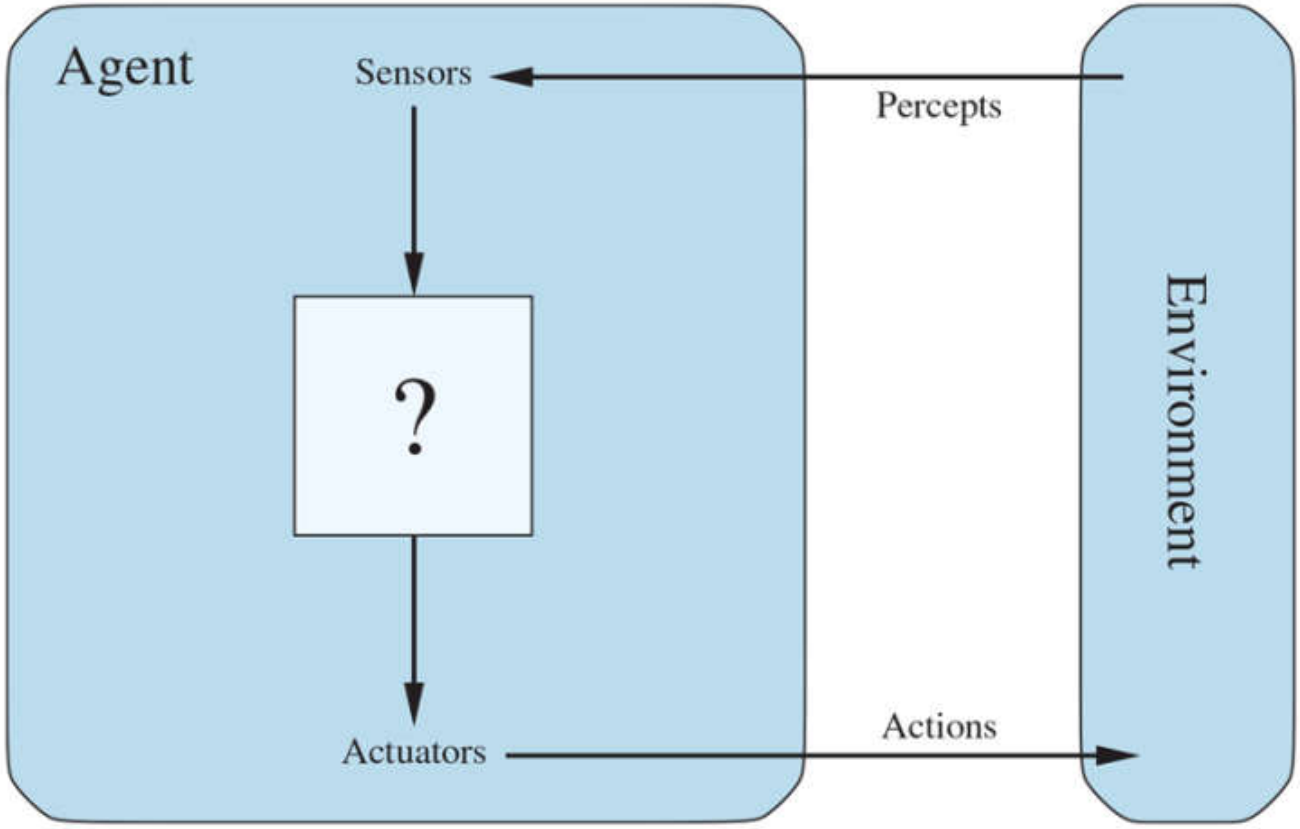
- Problem-solving agents
- Example problems
  (Toy problems & Real-world problems)
- Searching for solutions

PROBLEM SOLVING AGENTS

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- One kind of goal-based agent. (Goal-based agent considers future actions and the desirability of their outcomes.)
- Decides what to do by **finding sequences of actions** that lead to desirable states.
- Intelligent agents are supposed to maximize their performance measures.
- Achieving this is sometimes simplified if the agent can adopt a **goal** and aim at satisfying it.

Why and how an agent might do this?

PROBLEM SOLVING AGENTS

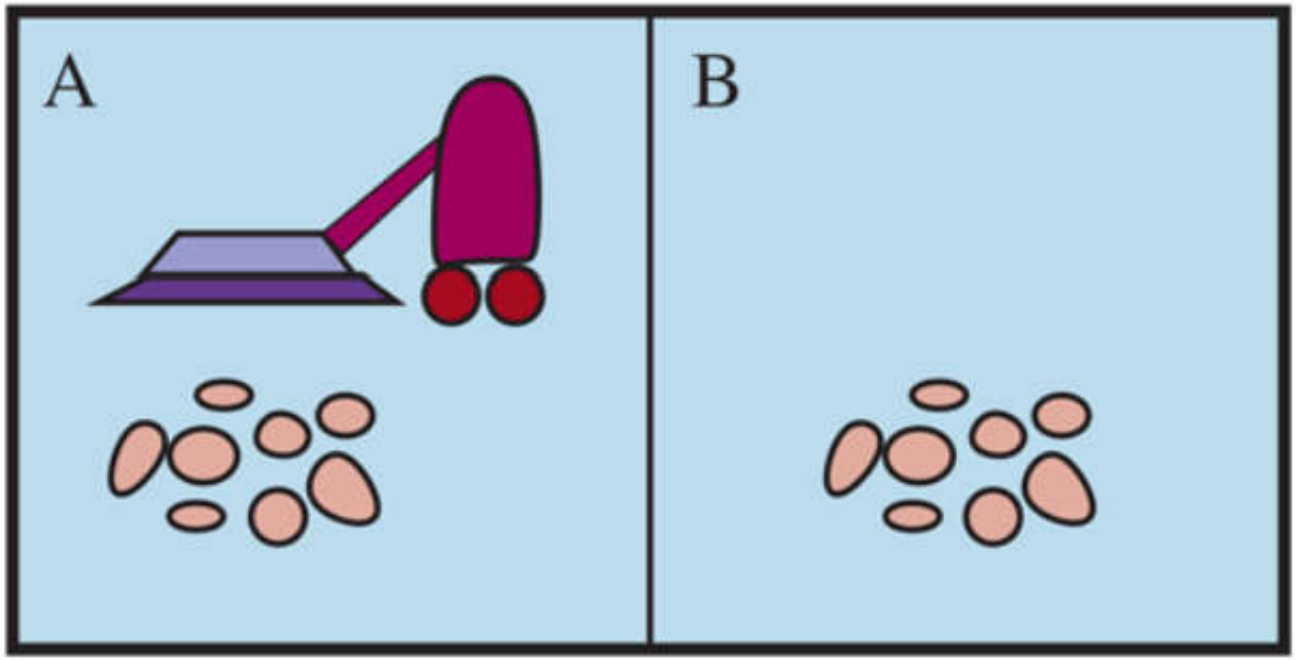


Agents interact with environments through sensors and actuators.

PROBLEM SOLVING AGENTS

- A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators
- A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.
- A software agent receives file contents, network packets, and human input (keyboard/mouse/touchscreen/voice) as sensory inputs and acts on the environment by writing files, sending network packets, and displaying information or generating sounds.
- The environment could be everything—the entire universe! In practice it is just that part of the universe whose state we care about when designing this agent—the part that affects what the agent perceives and that is affected by the agent's actions.

PROBLEM SOLVING AGENTS



A vacuum-cleaner world with just two locations. Each location can be clean or dirty, and the agent can move left or right and can clean the square that it occupies. Different versions of the vacuum world allow for different rules about what the agent can perceive, whether its actions always succeed, and so on.

Example (1/2)

- Suppose we have an agent in the City of Arad, Romania enjoying a touring holiday.

- With performance measures containing many factors:

  -Improve suntan

  -enjoy the nightlife (such as it is),

  -avoid hangovers

  ....

- The decision problem is a complex one with many tradeoffs.

Example (2/2)

- Now suppose the agent has a nonrefundable ticket to fly out of Bucharest the following day.

- Than it makes sense for the agent to adopt the **goal** of **getting to Bucharest**.

- By adopting this goal,
  - The agent rejects the actions that don't reach Bucharest on time.

  - Thus, the agent's decision problem is greatly simplified.

Road map of Romania

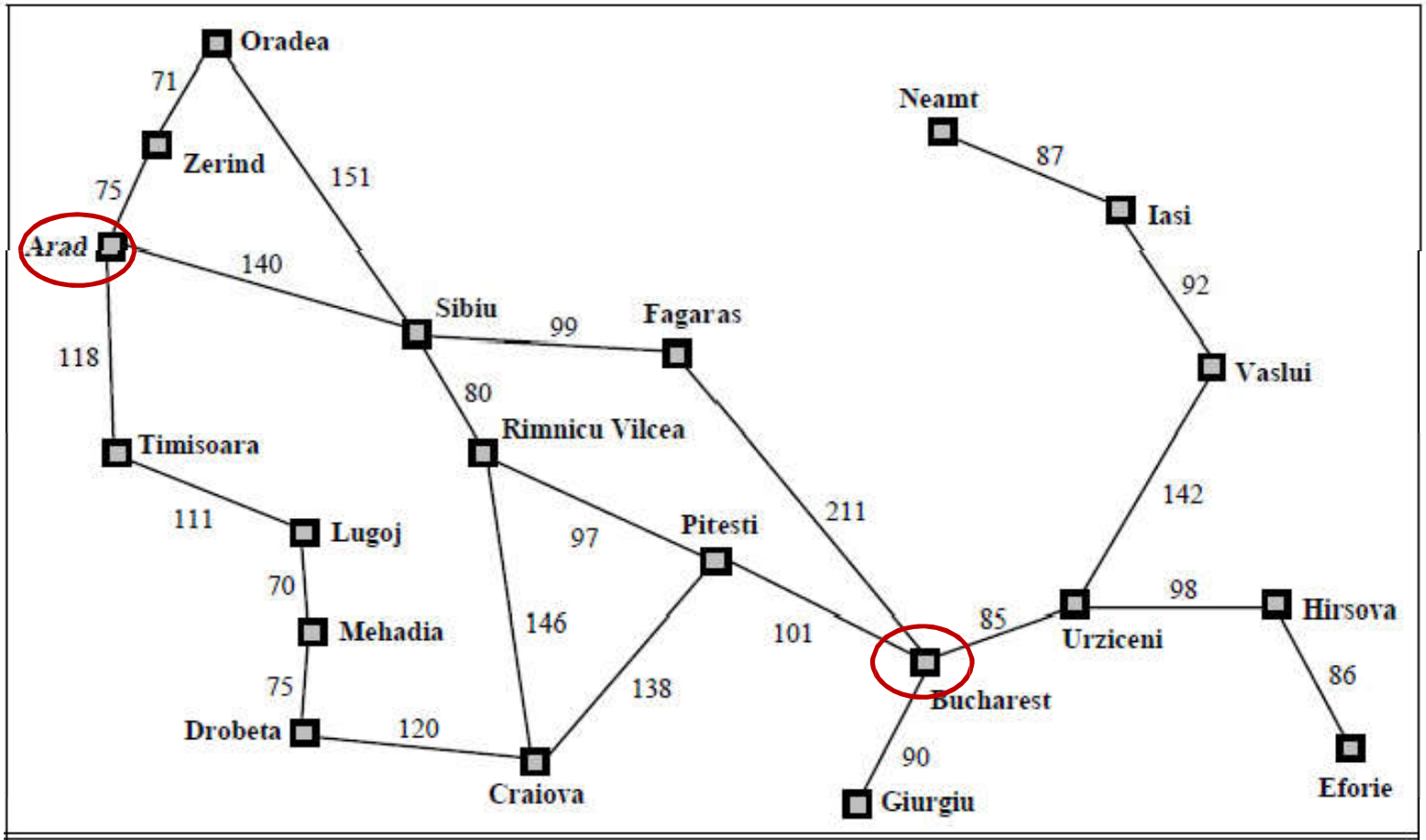


Figure 3.2 A simplified road map of part of Romania.

Goal formulation

- Goals help organize behavior by limiting the objectives that the agent is trying toachieve.

- The agent's task is to find out which sequence of actions will get it to a goalstate

- **Goal formulation** , based on the current situation and the agent's performance measure, is the first step in problem solving.

Problem formulation

- **Problem formulation:** the process of deciding what actions and states to consider, given a goal.

- in the example

-actions: driving from one major town to another

-states: being in a particular town

“Formulate, Search, Execute”

- The process of looking for a sequence that leads to a goal state is called **search**.
- A search algorithm takes a problem as input and returns a **solution** in the form of an action sequence.
- Once a solution is found, the actions it recommends can be carried out. This is called the **execution** phase.

- Thus, we have a simple “formulate, search, execute” design for the agent.

Assumptions for environment

- Assumptions for the **environment** of agent design in figure 3.2 are:

- **Static:** without paying attention to any changes that might be occurring in the environment.
- **Observable** "knows the initial state"
- **Discrete**: "alternative courses of action" can be enumerated
- **Deterministic**: No unexpected events

- All these assumptions mean that we are dealing with the easiest kinds of environments.

Well defined problems andsolutions

- **Goal formulation** is the first step in problem-solving.

- Then we need the **problem formulation** which is defined **by four components:**

-Initial state
-Successor function (or operator)
-Goal test
-Path cost

Components of a problem formulation

1-Initial state

The state that the agent starts in. -ex: in(Arad)

2-Successor function

-Given a particular state *x SUCCESSOR-FN(x)* returns a set of <action,successor> ordered pairs

-Ex: From state in(Arad) successor function returns:

{<Go(Sibiu),In(Sibiu)>,<Go(Zerind),In(Zerind)>,
Go(Timisoara),In(Timisoara)>}

Components of a problem formulation

3-Goal test

-Determines whether a given state *x* is a goal state.

"*x*=in(Bucharest)"

Goal states can be either

-explicit "in(Bucharest)"

-abstract property "checkmate" in chess

4-Path cost

-A function that assigns a numeric cost to each path.

-Sum of **step costs** c(x, a, y) >0

State space, path

- Together, the initial state and successor function implicitly define the **state space** of the problem
  -the set of all states reachable from the initial state.

- The state-space forms a graph in which the nodes are states and the arcs between nodes are actions.

- A **path** in the state space is a sequence of states connected by sequences of actions.

State space graph

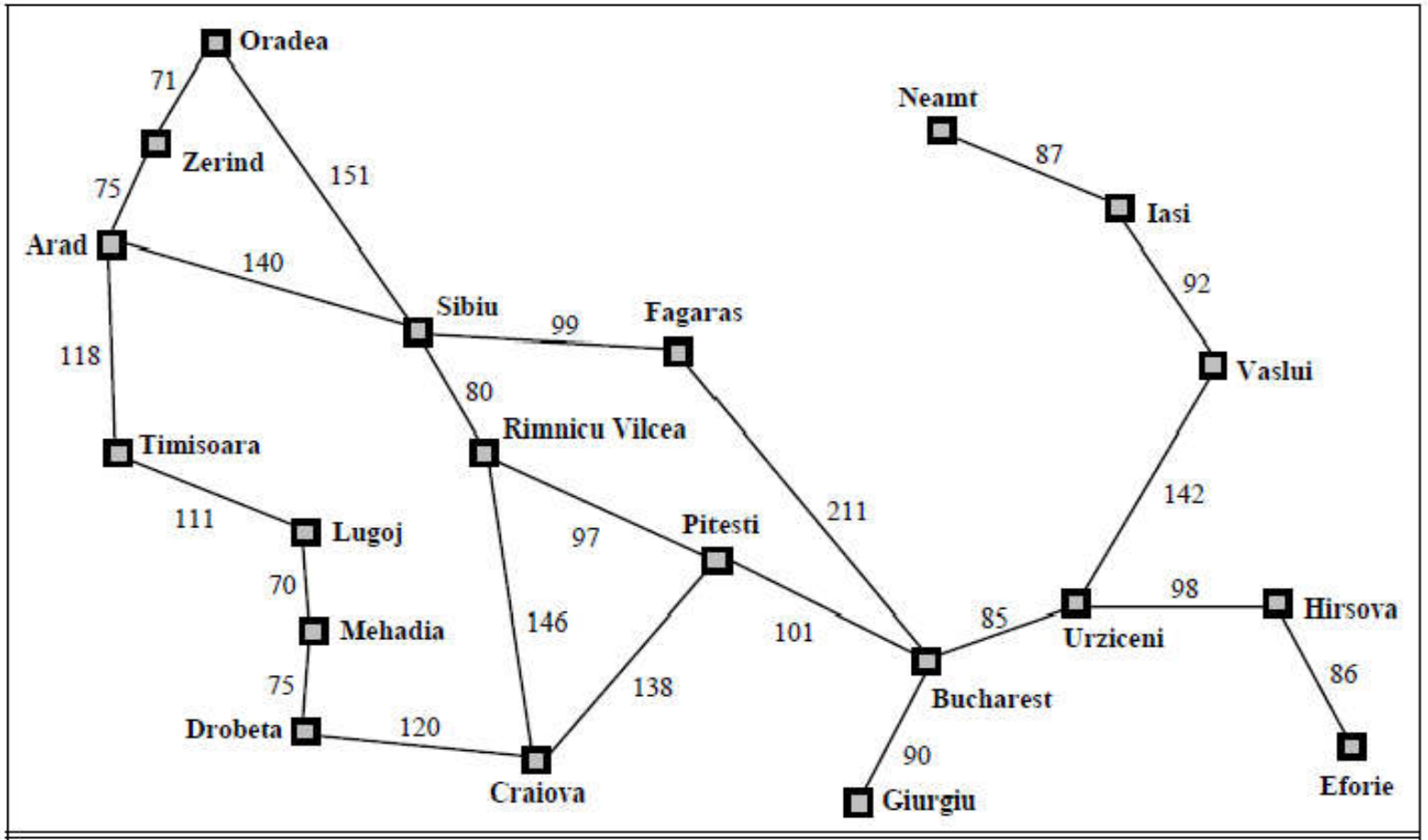


Figure 3.2 A simplified road map of part of Romania.

Solution quality, optimal solution

- A **solution** to a problem is a path from the initial state to a goal state.
- **Solution quality** is measured by the path cost function and an **optimal solution** has the lowest path cost among all solutions.

Formulating problems

- We previously proposed a formulation of the problem getting to Bucharest in terms of the initial state, successor function, goal test, and path cost.

- This formulation seems reasonable, yet it omits a great many aspects of the real world.

- A state of the real world includes **more detail** than the simple state description "in(Bucharest)".

(details like traveling companions, condition of the road, what is on the radio, the weather, etc.)

EXAMPLE PROBLEMS

- **Toy problem**

  -Intended to illustrate or exercise various problem-solving methods.

  -Concise, exact description

  -Can be used to compare the performance of algorithms

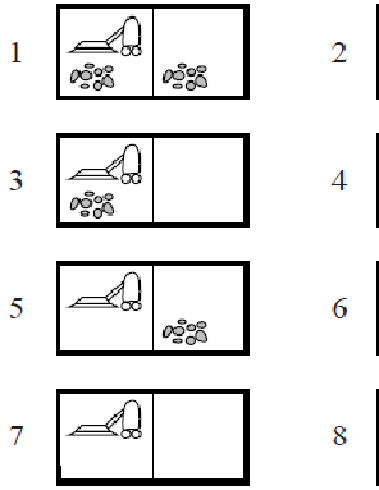
- **Real-world problem**

  -Whose solutions do people actually careabout

  -They tend not to have a single agreed-upon description

Some toy problems

Vacuum world



States: location of the agent, containing dirt or not. possible world states.

$2x2^2$

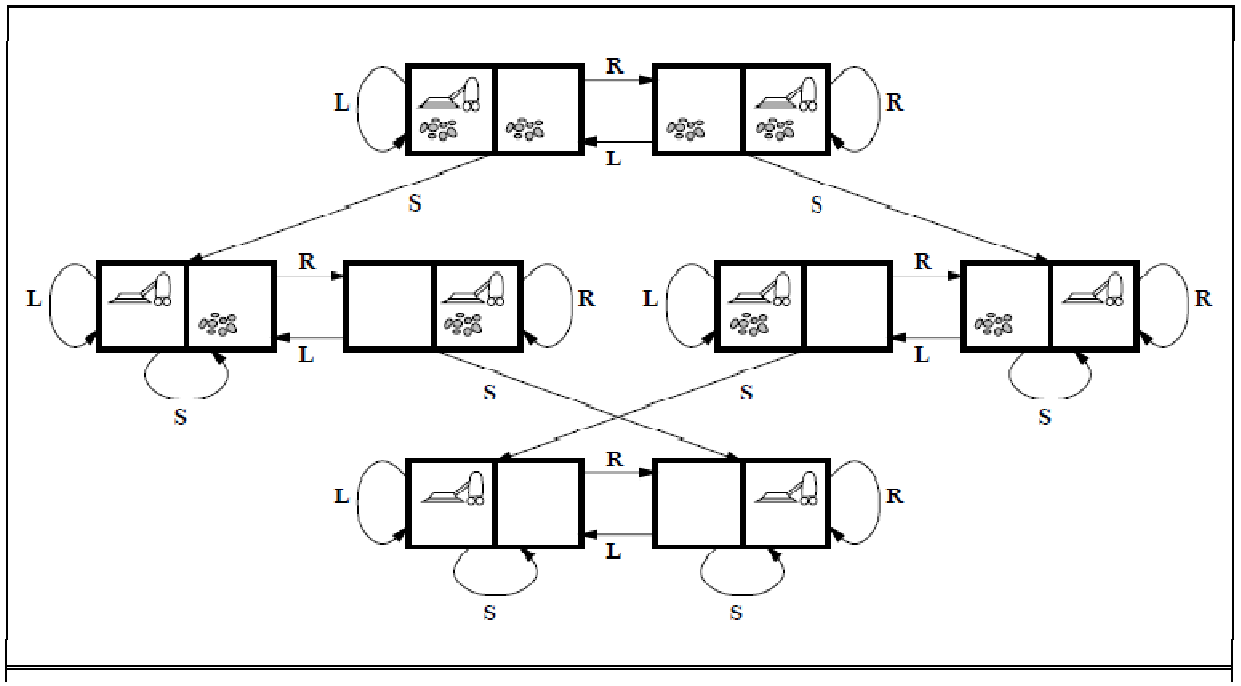
Initial state: any state

Successor function: legal states that result from three actions (Left Right and Suck)

Goal state: checks whether all the squares are clean.

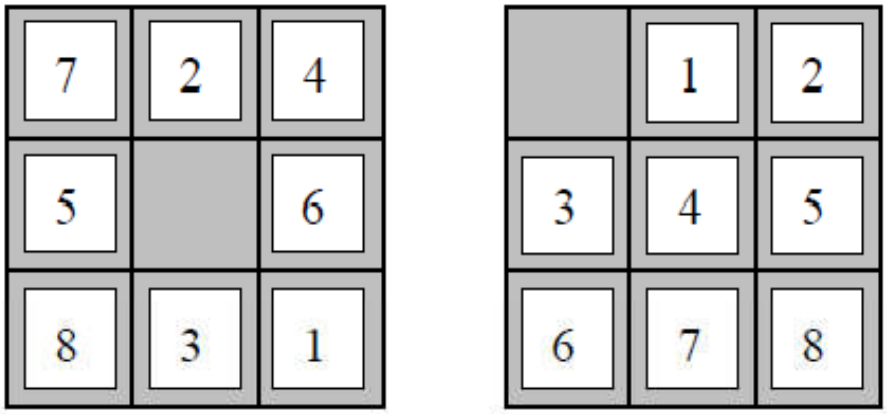
Path cost: # of steps in the path (each step costs 1)

Some toy problems



State space for the vacuum world

8-Puzzle



- Consist of a 3x3 board with 8 numbered tiles and a blank space.
- Tile adjacent to the blank space can slide into the space
- Object: reach a specified goal state

Formulation of 8-Puzzle

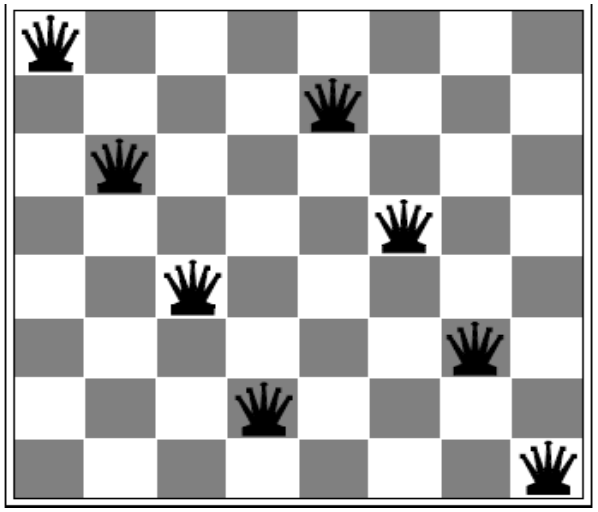
- **State space:** location of each of the eight tiles and the blank
- **Initial state:** Any state
- **Successor function:** states resulting from trying the four actions(blank moves left, right, up, or down)
- **Path cost:** number of steps in the path

- 8-puzzle belongs to the family of sliding block puzzles
- This general class is known to be NP-complete
- 9!/2=181,440 reachable states and easilysolved

8-Queens Problem

- Place 8-queens on a chessboard such that no queen attacks any other.

- Two main kinds of formulation



1- Complete-state formulation

Starts with all 8 queens on the board and moves them around.

8-Queens Problem

2-Incremental formulation:

-**States:** Any arrangement of 0 to 8 queens on the board

-**Initial state:** no queens on the board

-**Successor function:** Add a queen to any empty square

-**Goal test:** 8 queens are on the board, none attacked.

($64x63x \dots x57 \approx 3x10^4$ possible states to investigate.)

8-Queens Problem

Prohibit placing a queen in any square that is already attacked(a better formulation)

-**States:** Arrangements of n queens ($0 \leq n \leq 8$), one per column in the leftmost n columns, with no queen attacking another

-**Successor function:** add a queen to any square in the leftmost empty column such that it is not attacked by any other queen (Reduces state space from $3x10^4$ to just 2,057)

Real world problems

- **Some of the real-world problems are,**

-Route finding problem
-Touring problems
-Travelling salesperson problem
-VLSI layout problem
-Robot navigation
-Automatic assembly sequencing
-Internet searching

Route finding problem

- Used in a variety of applications

  -routing in computer networks, military operations planning, airline travel planning systems, etc.

- Typically, complex to specify

Route finding problem

- A simplified example of an airline travel problem:

  -**States:** Location (e.g., an airport) and the current time

  -**Initial state:** Specified by the problem

  -**Successor function:** States resulting from taking any scheduled flight (seat class and location), leaving later than the current time.

  -**Goal test:** Are we at the destination by some prespecified time?

  -**Path cost:** Depends on monetary cost, waiting time, flight time, seat quality, and so on.

  -many additional complications to handling byzantine structure that airlines impose

  -Contingency plans should also be included

Touring problems

- Closely related to route-finding problems
- Ex: **"Visit every city at least once, starting and ending at Bucharest"**

- Important differences,
- **States:** "current location, set of cities the agent has visited"

  -Ex: in Vaslui; visited{Bucharest, Urziceni, Vaslui}
- **Initial state:** "In Bucharest; visited(Bucharest)"
- **Goal test: -**Is the agent in Bucharest?

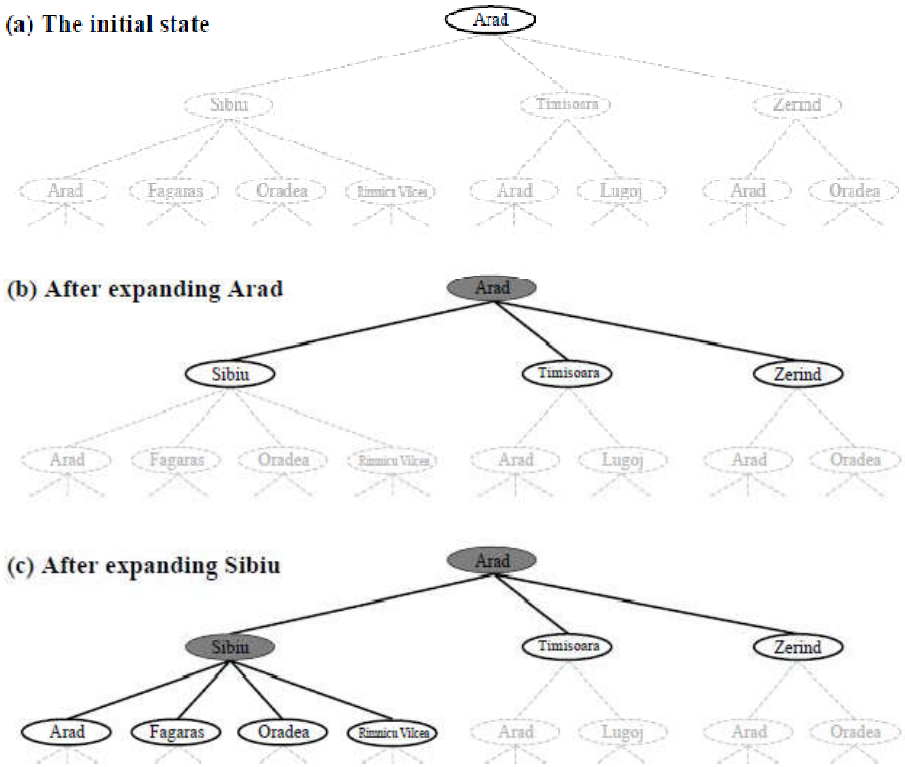
  -Visited all cities?

Traveling salesperson problem (TSP)

- A touring problem in which
  -Each city must be visited **exactly once**
- Aim: Find the shortest tour
- Knowns to be NP-hard

SEARCHING FOR SOLUTIONS

- Having formulated some problems, we now need to solve them.
- This is done by a search through the state space.
- This chapter deals with some search techniques that use an explicit **search tree.**



Partial search trees

Problem: find a route from Arad to Bucharest.

-Expanding : Applying successor function to the current state, thereby generating a new set of states.

-The choice of which state to expand is determined by the **search strategy**.

-When a goal state found, the solution is sequence of actions from goal node back to the root.

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

-Frontier: a queue that stores the set of nodes which are generated but not yet expanded

Representation of Nodes

- **A node is a data structure with five components**
  - **State**: the state in the state space that the node corresponds
  - **Parent-node**: the node in the search tree that generated this node
  - **Action**: the action that was applied to the parent to generate the node
  - **Path-cost**: cost of the path from the initial state to the node
  - **Depth**: The number of steps along the path from the root.

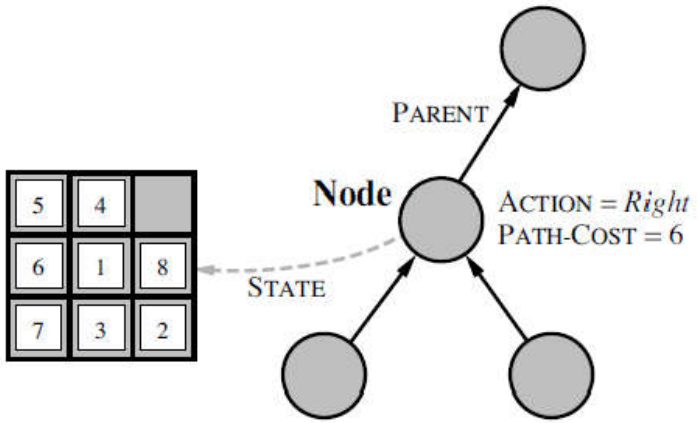


Figure 3.10 Nodes are the data structures from which the search tree is constructed. Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent.

-**Node**: bookkeeping data structure to represent a search tree

-**State**: corresponds to the configuration of the world

Measuring problem-solving performance

Evaluation of an algorithm's performance

- **Completeness:** Is the algorithm guaranteed to find a solution when there is one?
- **Optimality:** Does the strategy finds the optimal solution
- **Time complexity:** How long does it take to find a solution?
- **Space complexity:** How much memory is needed to perform the search?

Measuring problem-solving performance

- **Time** and **space** complexity are always considered concerning some measure of the problem difficulty.
- In AI, complexity is expressed in terms of three quantities:
  - B, the branching factor, max number of successors at any node
  - D, the depth of the shallowest goal node
  - M, the maximum length of any path in the state space
- **Time** is often measured in terms of several **nodes generated** during the search.
- **Space** is measured in terms of the **maximum number of nodes stored in memory**.

Assessing the efectiveness of an **algorithm**

- Search cost
  - depends on the time complexity
  - can also include a term for memory usage
- Total cost
  - combines the search cost and the path cost of the solution found

Conclusions

- Interesting problems can be cast as search problems, but you've got to set up state space
- Problem formulation usually requires abstracting away real-world details to defi a state space that can feasibly be explored