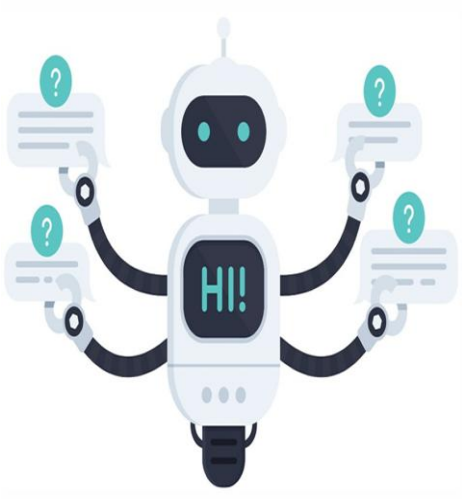
Solving Problems by Searching



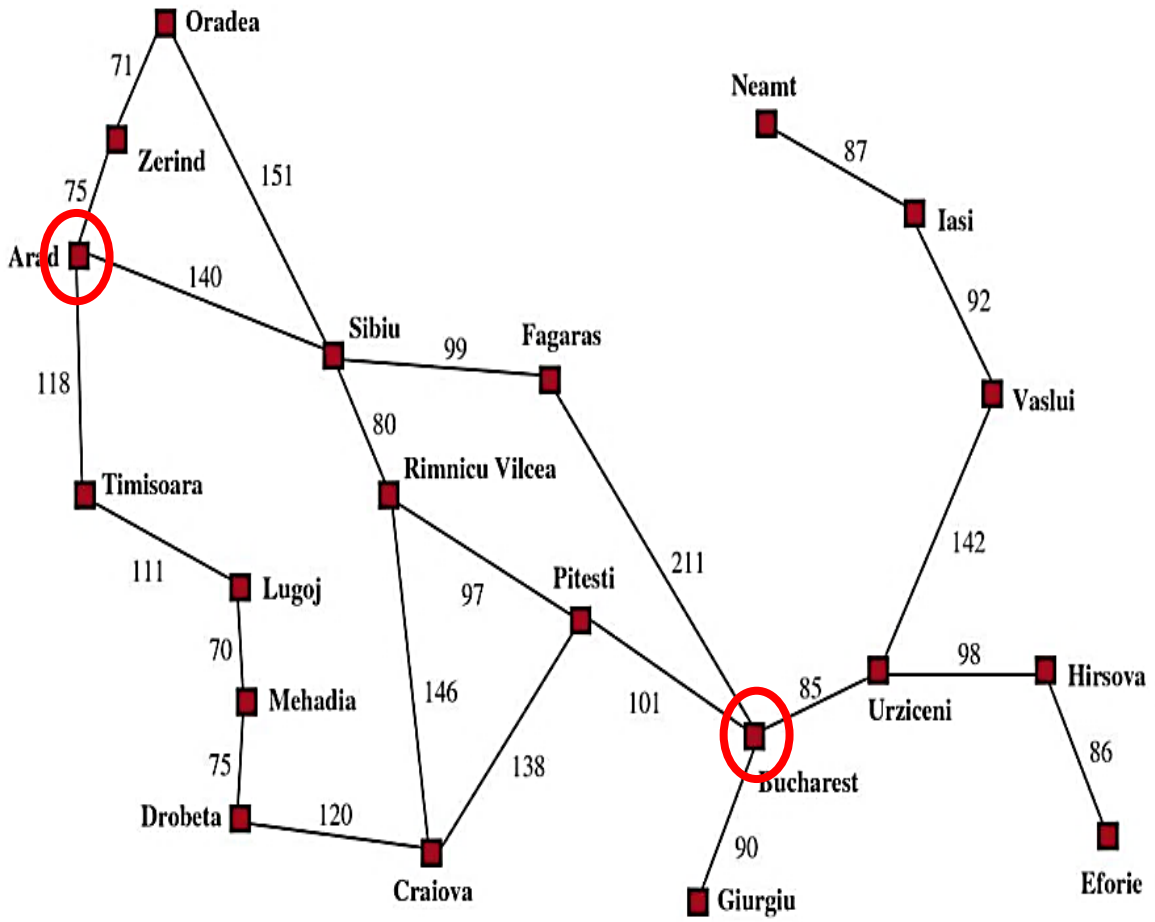
Prof. G. Rama Mohan Babu
Professor & HoD, CSE(AI&ML)
RVR&JCCE, Guntur - 19

3.1. Problem Solving Agents

- Problem Solving Agent - sequence of actions that find path to the *goal state.*
- These agents utilize various problem-solving techniques, algorithms, and heuristics to explore and evaluate potential solutions and then select the most appropriate course of action.
- The process of looking for a sequence of actions that reaches the goal 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.

Example: The Romania Problem (Trip to Bucharest)

- An agent is currently in the city of Arad, Romania for enjoying touring holiday.
- The agent has a nonrefundable ticket to fly out of Bucharest the following day.
- Goal: Reaching Bucharest on time.
- The agent observes street signs and sees that there are three roads leading out of Arad: one toward Sibiu, one to Timisoara, and one to Zerind



A simplified road map of part of Romania, with road distances in miles.

3.1.1. Well-defined problems and solutions

- A problem can be defined formally by five components:

1. The *initial state* that the agent starts in. *In(Arad)*
2. A description of possible *actions* available to the agent. Given a state *s* the function *ACTION(s)* returns a finite set of actions that can be executed in *s*.

   ACTIONS(Arad) = {Go(Sibiu), Go(Timisoara), Go(Zerind)}

3. A *transition model* – describes what each action does. The function *RESULT(s, a)* returns the state that results from doing action *a* in state *s*.

   RESULT(In(Arad), Go(Zerind)) = In(Zerind)

   - *Successor* – any state reachable from a given state by a single action.
   - *State space of problem* – the initial state, actions, and transition model. ie. the set of all states reachable from the initial state by any sequence actions.
   - *The state space form a directed network (graph) in which nodes are states and the links are actions.*

4. *The goal test* – which determines whether a given state is goal state or not. *Goal states = {In(Bucharest)}*
5. *The path* – is sequence of actions, and a solution is a path from the initial state to a goal state.
   - The path cost function – assigns a numeric cost for each path.
   - The total cost of a path is the sum of the individual action costs.

- *A solution to the problem is an action sequence that leads from the initial state to final state*.
- *Solution quality is measured by the path cost function*.
- *An optimal solution has the lowest path cost among all solutions*.

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: 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
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Figure 3.1 A simple problem-solving agent. It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over.

3.1.2. Formulating problems

- Problem formulation is the process of deciding what actions and states to consider, for given a goal.
- The process of removing detail from a representation is called abstraction. A good problem formulation has the right level of detail.
- There are so many details in the Romania problem.
- In the actual world includes condition of the road, the weather, the traffic, and so on.
- Left out all other details in the state space description because they are irrelevant to the problem of finding a route to Bucharest.

3.2. Example Problems

- A *toy problem* is intended to illustrate or exercise various problem solving methods. It can be given a concise, exact description and hence is usable by different researchers to compare the performance of algorithms.
  - Examples: Vacuum World, 8 Puzzle, 8-Queens etc.
- A *real-world problem* is one that has practical solutions people care about. These problems usually don't have a single description, but we can provide a general idea of their formulations.
  - Examples: Route Finding, Travelling Sales Person, VLSI design, Roboat Navigation, etc.

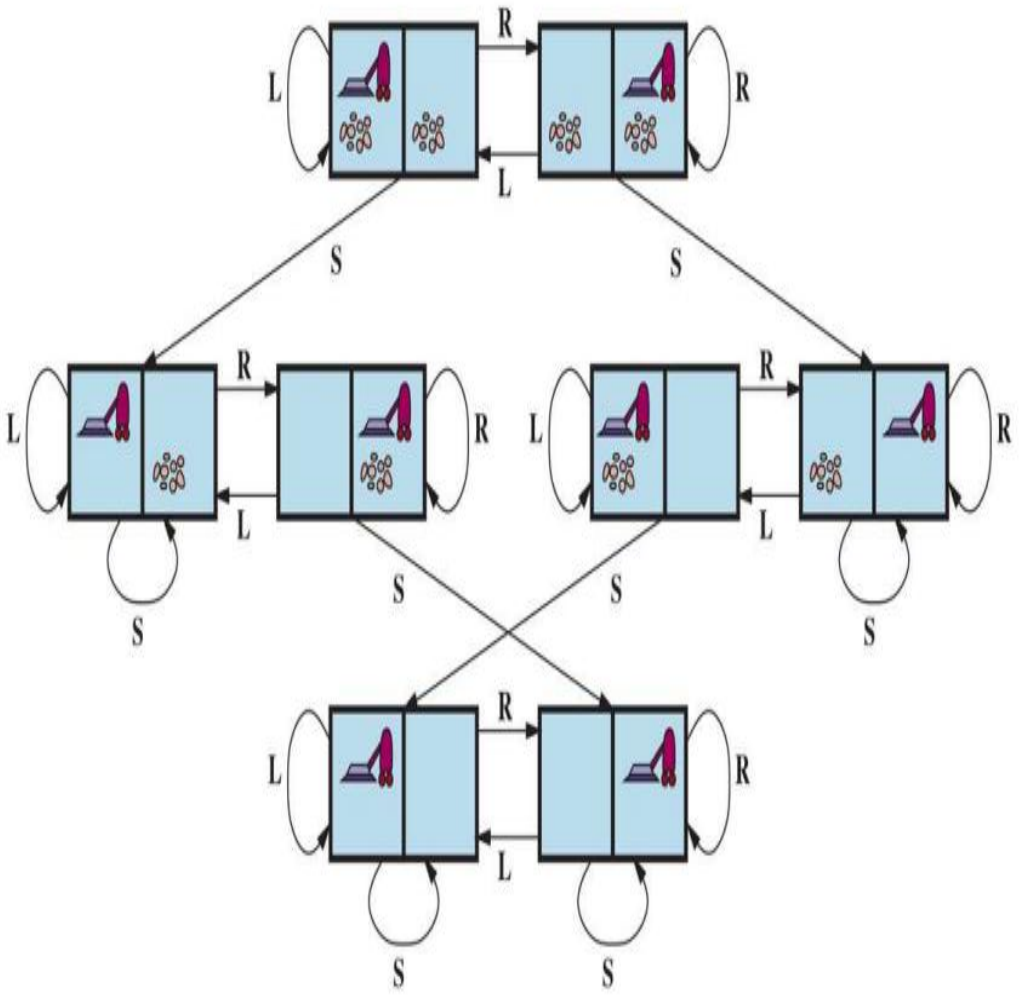
3.2.1. Toy Problems / Standardized problems

- Standard AI problems have clear, well-defined goals and parameters.
- A grid world problem is a two-dimensional rectangular array of square cells in which agents can move from cell to cell.
- Typically the agent can move to any obstacle-free adjacent cell horizontally or vertically and in some problems diagonally.
- Cells can contain objects, which the agent can pick up, push, or otherwise act upon; a wall or other impassible obstacle in a cell prevents an agent from moving into that cell.

Example 1: Vacuum World Problem/Toy Problem

– The vacuum world can be formulated as a grid world problem as follows:

- *STATES* – A state of the world says which objects are in which cells.
- For the vacuum world, the objects are the agent and any dirt.
- In the simple two-cell version, the agent can be in either of the two cells, and each call can either contain dirt or not, so there are $2 \times 2 \times 2 = 8$ states.
- In general, a vacuum environment with cells has states. In general, a vacuum environment with n cells has $n2^n$ states.



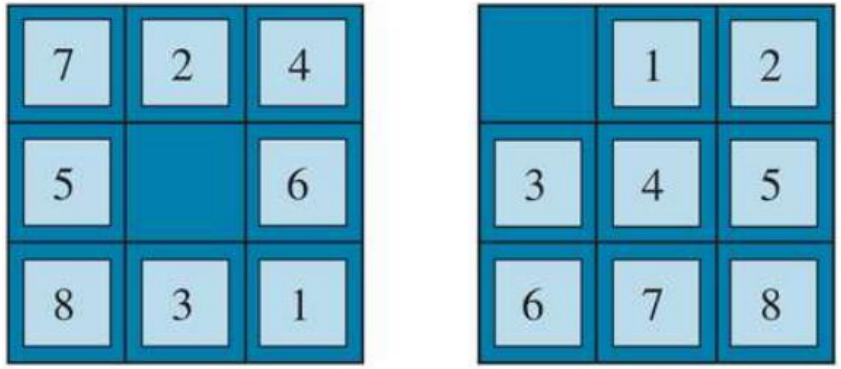
The state-space graph for the two-cell vacuum world. There are 8 states and three actions for each state: L = *Left*, R = *Right*, S = *Suck*.

1. *STATES* – A state of the world says which objects are in which cells.
2. *INITIAL STATE* – Any state can be designated as the initial state.
3. *ACTIONS* – In the two-cell world, possible actions are *Suck, move Left, and move Right*.
4. *TRANSITION MODEL* – the actions have expected effects, except the moving left in leftmost square, moving right in rightmost square, sucking in clear square have no effect.
5. *GOAL STATES* – The states in which every cell is clean.
6. *ACTION COST* – for each action costs 1, so the path cost is the number of steps in the path.

Example 2: 8 Puzzle Game/sliding-block Problem

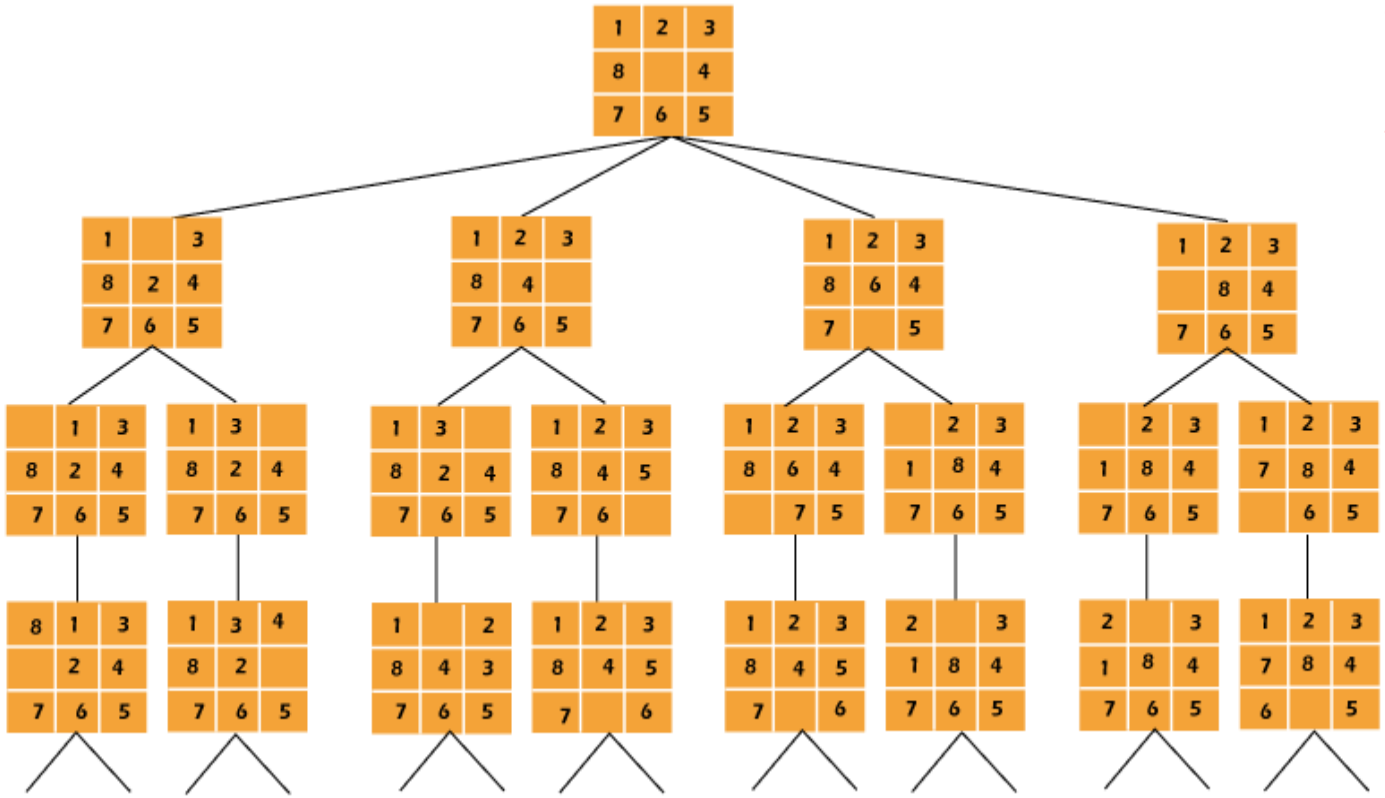
- *8-puzzle game consists of a 3×3 grid with eight numbered tiles and one blank space, and the 15-puzzle on a 4×4 grid. The object is to reach a specified goal state, such as the one shown in the figure.*

1. *STATES* – A state description specifies the location of each of the tiles.
2. *INITIAL STATE* – Any state can be designated as the initial state.
3. *ACTIONS* – the possible actions are to move the blank tile Left, Right, Up, or Down. If the blank is at an edge or corner then not all actions will be applicable.



Start State Goal State

4. *TRANSITION MODEL* – Maps a state and action to a resulting state; for example, if we apply Left to the start state, the resulting state has the 5 and the blank switched.
5. *GOAL STATE* – any state could be the goal, typically specify a state with the numbers in order.
6. *ACTION COST* – each action costs 1.



8-puzzle: 9! /2 = 181,440 solvable states.
15-puzzle: 16!/2 = 1.3 trillion states.
24-puzzle: 25!/2 =7.6 trillion states.

Example 3: 8-queens problem

- The goal of the 8-queens problem is to place eight queens on a chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal.)
- There are two main kinds of formulation.
- An *incremental formulation* involves operators that augment the state description, starting with an empty state; for the 8-queens problem, this means that each action adds a queen to the state.
- A *complete-state formulation* starts with all 8 queens on the board and moves them around.

- The first incremental formulation one might try is the following:
  1. *States:* Any arrangement of 0 to 8 queens on the board is a state.
  2. *Initial state:* No queens on the board.
  3. *Actions:* Add a queen to any empty square.
  4. *Transition model:* Returns the board with a queen added to the specified square.
  5. *Goal test:* 8 queens are on the board, none attacked.
  - In this formulation, we have $64 \times 63 \times \cdots \times 57 = 1.8 \times 10^{14}$ possible sequences to investigate.
- A better formulation would prohibit placing a queen in any square that is already attacked:
  - *States:* All possible arrangements of n queens ($0 \leq n \leq 8$), one per column in the leftmost n columns, with no queen attacking another.
  - *Actions:* Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.
  - This formulation reduces the 8-queens state space from 1.8×10^{14} to just 2,057 and solutions are easy to find.

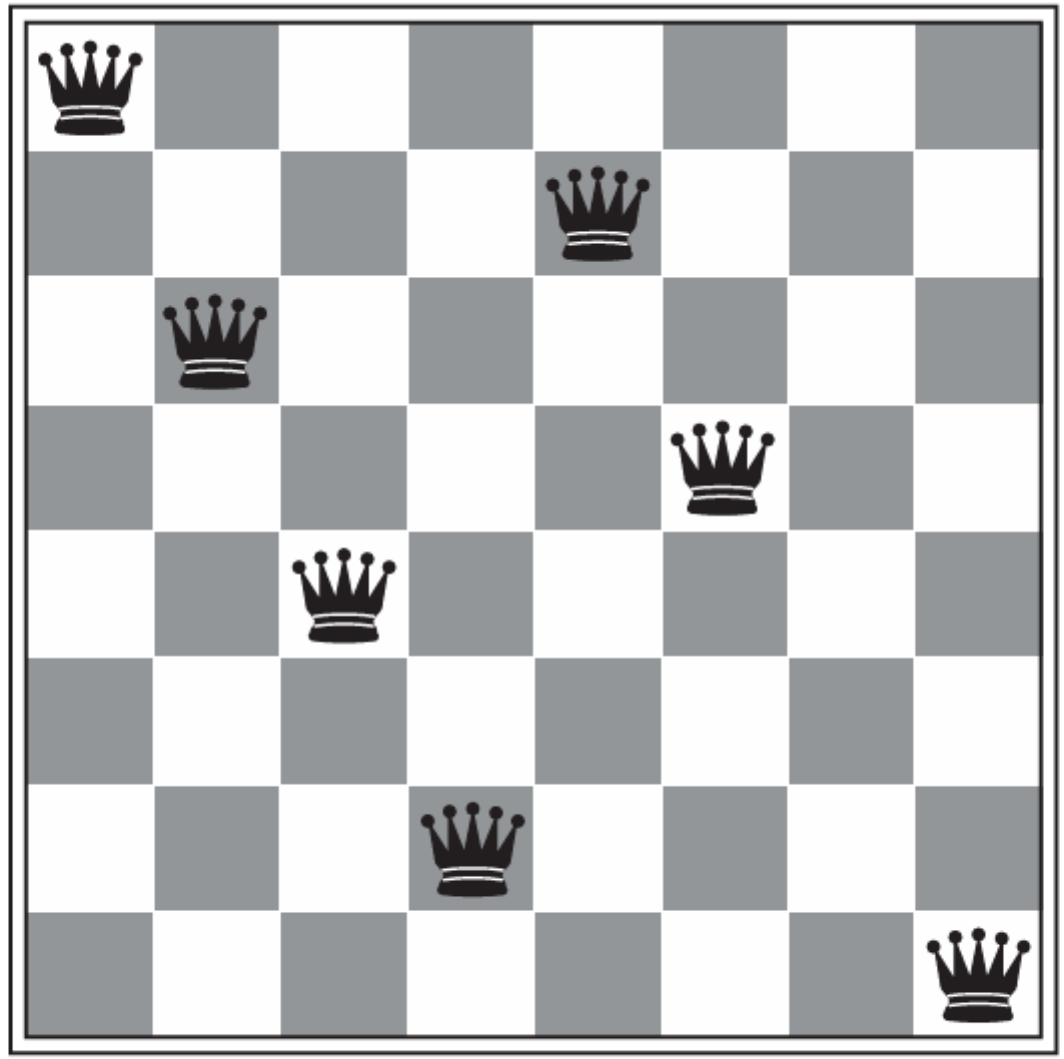


Figure 3.5 Almost a solution to the 8-queens problem. (Solution is left as an exercise.)

Example 4: Knuth's Conjecture / Infinite State Space

- Donald Knuth [1964] conjectured that, starting with the number 4, a sequence of factorial, square root, and floor operations will reach any desired positive integer. For example, we can reach 5 from 4 as follows:
- The problem definition is very simple:
  1. *States:* Positive numbers.
  2. *Initial state:* 4.
  3. *Actions:* Apply factorial, square root, or floor operation (factorial for integers only).
  4. *Transition model:* As given by the mathematical definitions of the operations.
  5. *Goal test:* State is the desired positive integer.

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}} \right\rfloor = 5 \, .$$

3.2.2. Real-world Problems

- Real-world problems are complex, involving multiple, often conflicting goals and parameters.
- *Example 1: Route Finding – Consider the airline travel problems that must be solved by a travel-planning Web site:*
    1. *STATES* – Each state includes a location (e.g., an airport) and the current time.
    2. *INITIAL STATE* – The user's home airport (specific to users query)
    3. *ACTIONS* – Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
    4. *TRANSITION MODEL* – The state resulting from taking a flight will have the flight's destination as the current location and the flight's arrival time as the current time.
    5. *GOAL STATE* – A destination city.
    6. *ACTION COST* – A combination of monetary cost, waiting time, flight time, etc.

- *Example 2: Touring Problem – Closely related to route-finding problems, but with an important difference.*
  - Each state must include not just the current location but also the set of cities the agent has visited.
  - So the initial state would be In(Bucharest), Visited({Bucharest})
  - A typical intermediate state would be In(Vaslui), Visited({Bucharest, Urziceni, Vaslui})
  - The goal test would check whether the agent is in Bucharest and all 20 cities have been visited.
- *Example 3: The traveling salesperson problem (TSP) is a touring problem in which each city must be visited exactly once. The aim is to find the shortest tour.*

- *Example 4: Robot navigation is a generalization of the route-finding problem described earlier. Rather than following a discrete set of routes, a robot can move in a continuous space with (in principle) an infinite set of possible actions and states.*
- *Example 5: AVLSI layout problem requires positioning millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield.*
- *Example 6: Automatic assembly sequencing - the aim is to find an order in which to assemble the parts of some object. If the wrong order is chosen, there will be no way to add some part later in the sequence without* undoing some of the work already done.