1.3.1 Problem Solving by Search

An important aspect of intelligence is ***goal-based*** problem solving.

The **solution** of many **problems can** be described by finding a **sequence of actions** that lead to a desirable **goal.** Each action changes the ***state*** and the aim is to find the sequence of actions and states that lead from the initial (start) state to a final (goal) state.

A well-defined problem can be described by:

- **Initial state**
- **Operator or successor function** - for any state x returns s(x), the set of states reachable from x with one action
- **State space** - all states reachable from initial by any sequence of actions
- **Path** - sequence through state space
- **Path cost** - function that assigns a cost to a path. Cost of a path is the sum of costs of individual actions along the path
- **Goal test** - test to determine if at goal state

What is Search?

Search is the systematic examination of **states** to find path from the **start/root state** to the **goal state.**

The set of possible states, together with *operators* defining their connectivity constitute the *search space*.

The output of a search algorithm is a solution, that is, a path from the initial state to a state that satisfies the goal test.

Problem-solving agents

A Problem-solving agent is a **goal-based** agent. It decides what to do by finding sequence of actions that lead to desirable states. The agent can adopt a goal and aim at satisfying it. To illustrate the agent's behavior, let us take an example where our agent is in the city of Arad, which is in Romania. The agent has to adopt a **goal** of getting to Bucharest.

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

The agent's task is to find out which sequence of actions will get to a goal state.

Problem formulation is the process of deciding what actions and states to consider given a goal.

Example: Route finding problem

Referring to figure 1.19

On holiday in Romania: currently in Arad.

Flight leaves tomorrow from Bucharest

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

Problem formulation
A **problem** is defined by four items: **initial state** e.g., —at Arad" **successor function** S(x) = set of action-state pairs e.g., S(Arad) = {[Arad -> Zerind;Zerind],….} **goal test**, can be explicit(স্পষ্ট), e.g., x = at Bucharest" implicit(উহ্য), e.g., NoDirt(x) **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.
Figure 1.17 Goal formulation and problem formulation

Search

An agent with several immediate options of unknown value can decide what to do by examining different possible sequences of actions that leads to the states of known value, and then choosing the best sequence. The process of looking for sequences actions from the current state to reach the goal state is called **search.**

The **search algorithm** takes a **problem** as **input** and returns a **solution** in the form of **action sequence.** Once a solution is found, the **execution phase** consists of carrying out the recommended action..

Figure 1.18 shows a simple —formulate, search, execute design for the agent. Once solution has been executed,the agent will formulate a new goal.

function SIMPLE-PROBLEM-SOLVING-AGENT(*percept*) **returns** an action
inputs : *percept*, a percept
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*

Figure 1.18 A Simple problem solving agent. It first formulates a **goal** and a **problem**,searches for a sequence of actions that would solve a problem,and executes the actions one at a time.

- The agent design assumes the Environment is
    - **Static** : The entire process carried out without paying attention to changes that might be occurring in the environment.
    - **Observable :** The initial state is known and the agent's sensor detects all aspects that are relevant to the choice of action
    - **Discrete :** With respect to the state of the environment and percepts and actions so that alternate courses of action can be taken
    - **Deterministic :** The next state of the environment is completely determined by the current state and the actions executed by the agent. Solutions to the problem are single sequence of actions

An agent carries out its plan with eye closed. This is called an open loop system because ignoring the percepts breaks the loop between the agent and the environment.

1.3.1.1 Well-defined problems and solutions

A **problem** can be formally defined by **four components**:

- The **initial state** that the agent starts in. The initial state for our agent of example problem is described by *In(Arad)*
- A **Successor Function** returns the possible **actions** available to the agent. Given a state x,SUCCESSOR-FN(x) returns a set of {action,successor} ordered pairs where each action is one of the legal actions in state x,and each successor is a state that can be reached from x by applying the action.

For example,from the state In(Arad),the successor function for the Romania problem would return

{ [Go(Sibiu),In(Sibiu)],[Go(Timisoara),In(Timisoara)],[Go(Zerind),In(Zerind)] }

- **State Space** : 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 a sequence of actions.
- Thr **goal test** determines whether the given state is a goal state.

- A **path cost** function assigns numeric cost to each action. For the Romania problem the cost of path might be its length in kilometers.
- The **step cost** of taking action a to go from state x to state y is denoted by c(x,a,y). The step cost for Romania are shown in figure 1.18. It is assumed that the step costs are non negative.
- A **solution** to the problem is a path from the initial state to a goal state.
- An **optimal solution** has the lowest path cost among all solutions.

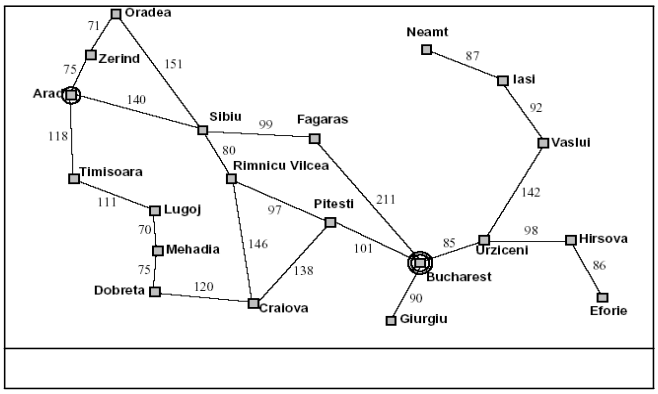

Figure 1.19 A simplified Road Map of part of Romania

1.3.2 EXAMPLE PROBLEMS

The problem solving approach has been applied to a vast array of task environments. Some best known problems are summarized below. They are distinguished as toy or real-world problems

A **toy problem** is intended to illustrate various problem solving methods. It can be easily used by different researchers to compare the performance of algorithms.

A **real world problem** is one whose solutions people actually care about.

1.3.2.1 TOY PROBLEMS

Vacuum World Example

- **States**: The agent is in one of two locations.,each of which might or might not contain dirt.
  Thus there are $2 \times 2^2 = 8$ possible world states.
- **Initial state**: Any state can be designated as initial state.
- **Successor function** : This generates the legal states that results from trying the three actions
  (left, right, suck). The complete state space is shown in figure 2.3
- **Goal Test** : This tests whether all the squares are clean.
- **Path test** : Each step costs one ,so that the the path cost is the number of steps in the path.

Vacuum World State Space

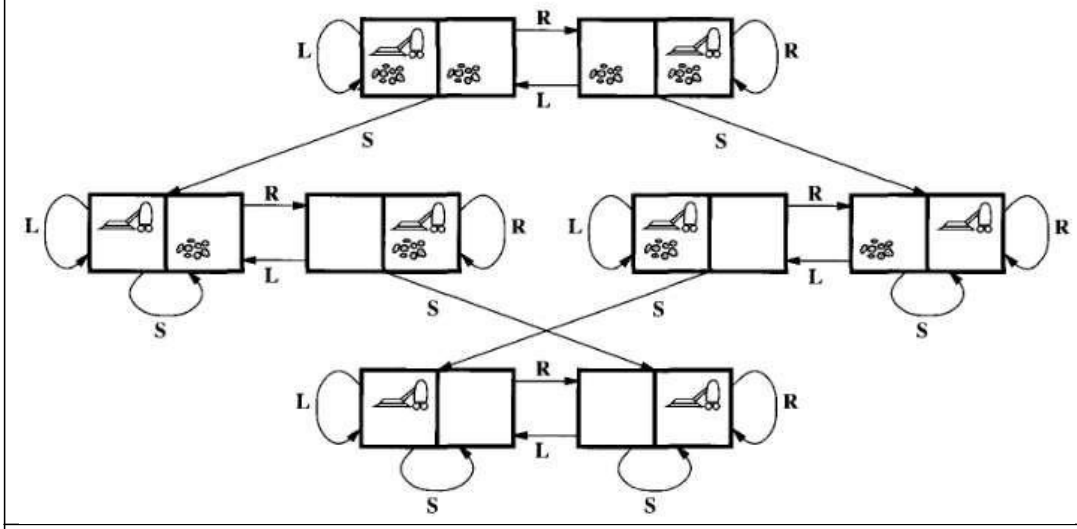

Figure 1.20 The state space for the vacuum world.
Arcs denote actions: L = Left,R = Right,S = Suck