SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)
Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai
Accredited by NAAC-UGC with 'A++' Grade (Cycle III) &
Accredited by NBA (B.E - CSE, EEE, ECE, Mech & B.Tech.IT)
COIMBATORE-641 035, TAMIL NADU

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

PROBLEM SOLIVING METHODS

Problem solving Methods – Search Strategies- Uninformed – Informed – Heuristics – Local Search Algorithms and Optimization Problems - Searching with Partial Observations – Constraint Satisfaction Problems – Constraint Propagation - Backtracking Search – Game Playing – Optimal Decisions in Games – Alpha – Beta Pruning – Stochastic Games

2.1 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 decide 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 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, e.g., x = at Bucharest" implicit, 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.
Goal formulation and problem formulation

2.2 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.

2.3 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 x 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
- **Goal Test**: This tests whether all the squares are clean.
- **Path test**: Each step costs one, so that the path cost is the number of steps in the path. **Vacuum World State Space**

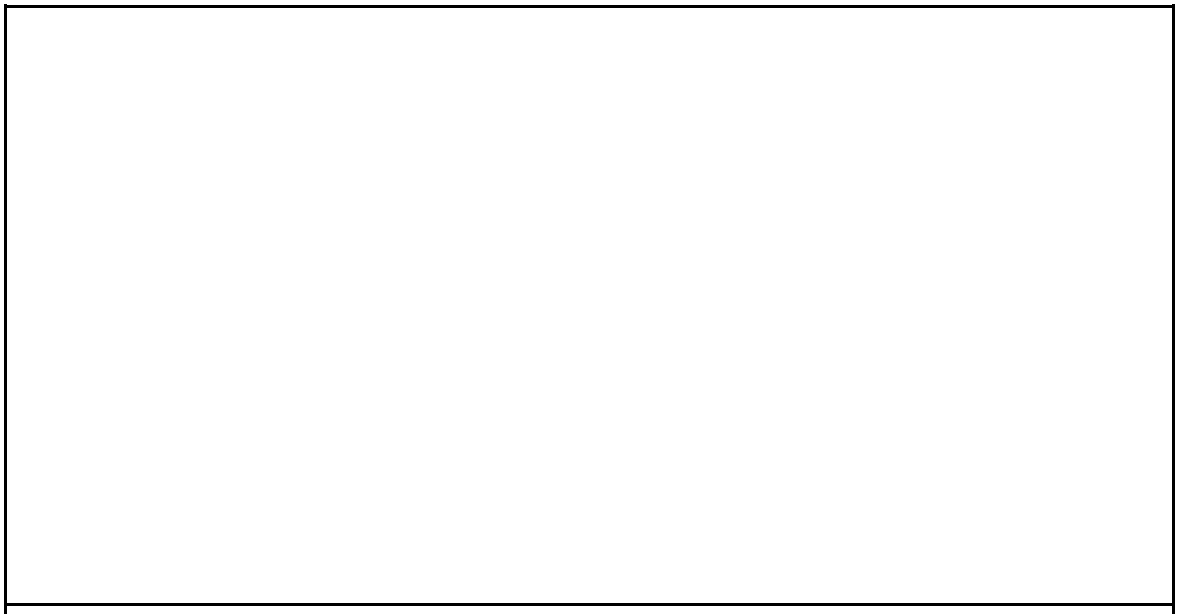

Figure 2.1 The state space for the vacuum world.
Arcs denote actions: L = Left, R = Right

The 8-puzzle

An 8-puzzle consists of a 3x3 board with eight numbered tiles and a blank

space. A tile adjacent to the balank space can slide into the space. The object is to reach the goal state, as shown in Figure 2.4

Example: The 8-puzzle

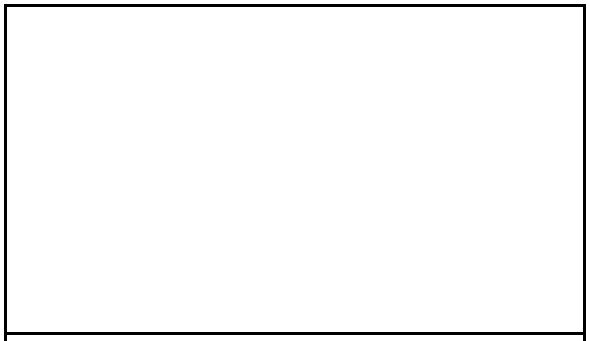

Figure 2.2 A typical instance of 8-puzzle

The problem formulation is as follows:

- **States** : A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
- **Initial state** : Any state can be designated as the initial state. It can be noted that any given goal can be reached from exactly half of the possible initial states.
- **Successor function** : This generates the legal states that result from trying the four actions(blank moves Left, Right, Up or down).
- **Goal Test** : This checks whether the state matches the goal configuration shown in Figure(Other goal configurations are possible)
- **Path cost** : Each step costs 1,so the path cost is the number of steps in the path.

The 8-puzzle belongs to the family **of sliding-block puzzles**, which are often used as test problems for new search algorithms in AI. This general class is known as NP-complete. The **8-puzzle** has 9!/2 = 181,440 reachable states and is easily solved.

The **15 puzzle** (4 x 4 board) has around 1.3 trillion states, an the random instances can be solved optimally in few milli seconds by the best search algorithms.

The **24-puzzle** (on a 5 x 5 board) has around 10^{25} states and random

instances are still quite difficult to solve optimally with current machines and algorithms.

8-Queens problem

The goal of 8-queens problem is to place 8 queens on the chessboard such that no queen attacks any other.(A queen attacks any piece in the same row, column or diagonal).

Figure 2.3 shows an attempted solution that fails: the queen in the right most column is attacked by the queen at the top left.

An **Incremental formulation** involves operators that augments the state description, starting with an empty state. For 8-queens problem, this means each action adds a queen to the state. A **complete-state formulation** starts with all 8 queens on the board and move them around. In either case the path cost is of no interest because only the final state counts.

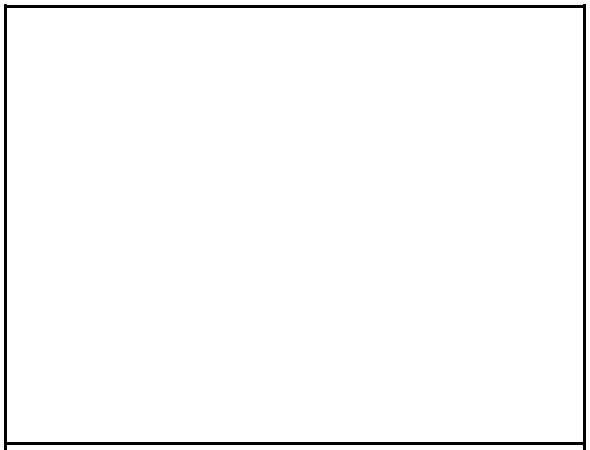

Figure 2.3 8-queens problem

The first incremental formulation one might try is the following:

- **States**: Any arrangement of 0 to 8 queens on board is a state.
- **Initial state**: No queen on the board.
- **Successor function**: Add a queen to any empty square.
- **Goal Test**: 8 queens are on the board, none attacked.

In this formulation, we have $64.63\ldots57 = 3 \times 10^{14}$ possible sequences to investigate.

A better formulation would prohibit placing a queen in any square that is already attacked.

- **States** : Arrangements of n queens (0 <= n < = 8),one per column in the left most columns, with no queen attacking another are states.

- **Successor function** : Add a queen to any square in the left most empty column such that it is not attacked by any other queen.

This formulation reduces the 8-queen state space from 3 x 10^{14} to just 2057,and solutions are easy to find.

For the 100 queens the initial formulation has roughly 10^{400} states whereas the improved formulation has about 10^{52} states. This is a huge reduction, but the improved state space is still too big for the algorithms to handle.

2.4 REAL-WORLD PROBLEMS

ROUTE-FINDING PROBLEM

Route-finding problem is defined in terms of specified locations and transitions along links between them. Route-finding algorithms are used in a variety of applications, such as routing in computer networks, military operations planning, and airline travel planning systems.

2.5 AIRLINE TRAVEL PROBLEM

The **airline travel problem** is specifies as follows**:**

- **States:** Each is represented by a location (e.g., an airport) and the

current time. - **Initial state:** This is specified by the problem.

- **Successor function:** This returns the states resulting from taking any scheduled flight (further specified by seat class and location),leaving later than the current time plus the within-airport transit time, from the current airport to another.

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

- **Path cost:** This depends upon the monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of date, type of air plane, frequent-flyer mileage awards, and so on.

2.6 TOURING PROBLEMS

Touring problems are closely related to route-finding problems, but with an important difference. Consider for example, the problem, “Visit every city at least once” as shown in Romania map.

As with route-finding the actions correspond to trips between adjacent cities. The state space, however, is quite different.

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.