2 Problems and State Space Search

Syllabus

Problems, State Space Search & Heuristic Search Techniques : *Defining The Problems As A State Space Search, Production Systems, Production Characteristics, Production System Characteristics, And Issues In The Design Of Search Programs, Additional Problems.*

Contents

2.1 Problem Defining and Solving Problem

GTU : Summer-13,15, Winter-15

2.1.1 What is Problem Solving ?

- For solving any type problem (task) in real world one needs formal description of the problem.
- One should have clear understanding of following aspects of the problem ⇒

1. What is the Explicit Goal of the Problem

- Goals help to organize behaviour of systems by limiting the objectives that the agent is trying to achieve. Goal formulation is based on the current situation and the agent's performance measure. It is first step towards problem solving.

2. What is Implicit Criteria for Success

- That is how success is defined. That will be the ultimate thing system needs to achieve, which is the problem solution's output.

3. What is the Initial Situation

- It means that what is going to be the start state of problem being solved.

4. Ability to Perform

- It tells how agents transforms from one situation to another, how operations and rules are specified which change the states of the problem during solution process.

2.1.2 Well Defined Problems

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

A problem can be defined formally by four components.

1) Initial state that the agent starts in

For example -

- Consider a agent program **Indian Traveller** developed for travelling from Pune to Chennai travelling through different states. The initial state for this agent can be described as **In (Pune)**.

2) A description of the possible actions available to the agent

- The most common formulation uses a successor funtion. Given a perticular state x, SUCCESSOR Function (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 (Pune)**, the successor function for Indian Traveller problem would return.

{
< Go (Mumbai), In (Mumbai) >
< Go (AhemdNagar), In (AhemdNagar)>
< Go (Solapur), In (Solapur)>
< Go (Satara), In (Satara)>
}

- Together, the initial state and successor function implicitly define the state space of the problem - which is 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 is the state space is a sequence of states connected by a sequence of actions.

3) The goal test, which determines whether a given state is goal (final) state. In some problems we can explicitely specify a set of goals. If a particular state is reached we can check it with set of goals and if a match is found success can be announced.

For example :

In Indian Traveller problem the goal is to reach chennai i.e. it is a singleton set **{In (Chennai) }**.

In certain types of problems we can not specify goals explicitly. Instead, goal is specified by an abstract property rather than an explicitly enumerated set of states.

For example :

In chess, the goal is to reach a state called "Checkmate" where the opponent's king is under attack and can not escape. This "Checkmate" situation can be represented using various state spaces.

4) A path cost function that assigns a numeric cost (value) to each path. The problem-solving **agent is expected to choose a cost-function that reflects its own performance measure.**

For Indian-Traveller agent we can have time requireded as cost for path-cost function. It should consider length of each road being travelled.

In general step-cost of taking action 'a' to go from state x to state y is denoted by c (x, a, y).

The above 4 elements define a problem and can be put together in single data structure which can be given as input to a problem-solving algorithm.

A solution to the problem is a path from the initial state to a goal state.

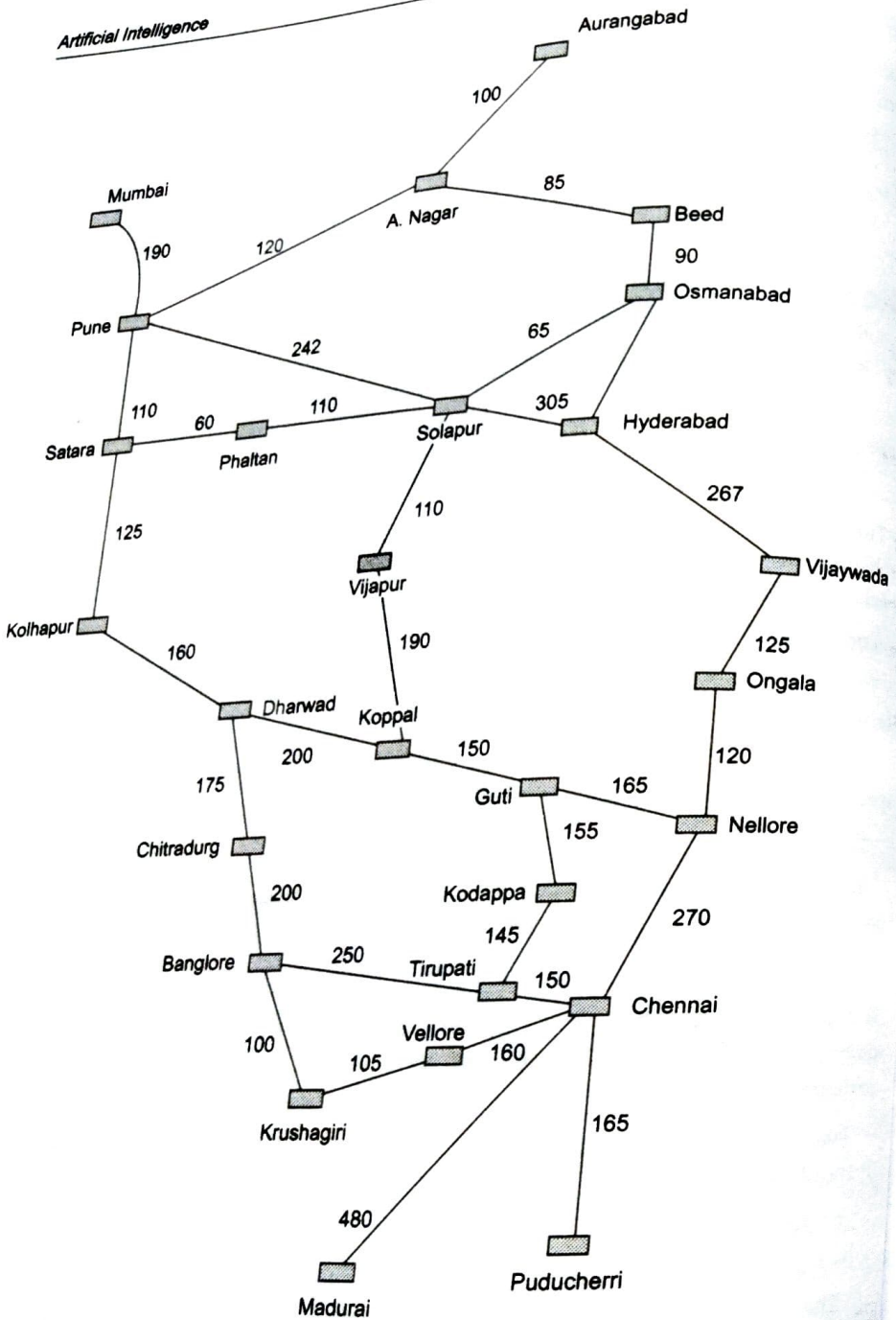


Fig. 2.1.1 Indian traveller map

We can measure quality of solution by the path cost function. We can have multiple solutions to the problem. The optimal solution will be the one with lowest path cost among all the solutions.

2.1.3 Problem Formulation Types

There are two main kinds of problem formulation ⇒

1) Incremental formulation

2) Complete-state formulation.

Depending upon problem requirements and specification one can decide which one to go for.

1) Incremental formulation

- It involves operators that augment the state description, starting with an empty state.
- It generates many sequences.
- Memory requirements is less as all states are not explored (exploration will be done till the goal is found).

For example -

For the 8-queens problem, incremental formulation states that, each action adds a queen to the state. In this formulation we have 64.63 ... 57 $\approx 3\times10^{14}$ possible sequences to investigate.

2) Complete state formulation

- In this initially we will have some basic configuration represented in initial state.
- Here while doing any action first the conditions on the actions will be checked so that the configuration state after the action will be same legal state.
- It takes up large memory as complete state space is generated. This formulation reduces number of sequences generated.

For example -

In 8-queen problem initially all the queens will be arranged on the board. The action will be 'move a queen to the next square such that it is not attacking'.

This complete state formulation reduces state space from 3×10^{14} (which is for incremental formulation) to just 2,057 and solutions are easy to find.

Example of Incremental formulation and complete-state formulation :

Consider 8-queen problem,

- **Incremental formulation**

1) States
   - Arrangement of upto 8 queens on the board.
2) Initial state
   - Empty board.
3) Successor function (operators)
   - Add a queen to any square.
4) Goal test
   - All queens on board
   - No queen attacked.

Properties : 3×10^{14} possible sequences.

- **Complete state formulation**

1) States
   - Arrangement of 8-queens on the board.
2) Initial state
   - All 8 queens on board.
3) Successor function (operators)
   - Move a queen to a different square.
4) Goal test
   - No queen attacked.

Properties : Good strategies can reduce the number of possible sequences which are considerable.

2.1.4 Solving the Problem

Finding the solution of a problem is procedure which involves following phases ←

1) Problem definition : Where in detailed specification of inputs and what constitutes an acceptable solution is described.

2) Problem analysis : Where in problem is studied through various view points like inputs, to the problem, environment of the problem, expected outputs.

3) Knowledge representation : Where in the known data about the problem and various expected stimuli from environment is represented in perticular format which is helpful for taking actions.

4) Problem solving : Where in the selection of best suited techniques for problem solutions are thought of and finalized.

2.1.5 Problem Solving Agents

2.1.5.1 Approach of Problem Solving Agent

- Goal based agents are also called as problem solving agent.
- Problem solving agent adapt to the task environment understand goal and achieve success -

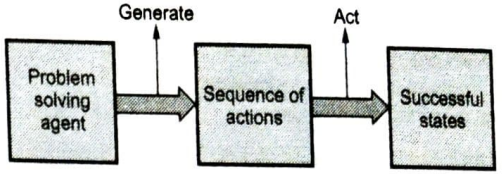


Fig. 2.1.2 Steps in problem solving

- Problem solving agents determine sequence of actions which generate successful state.
- Problem solving agent can be aimed at maximizing performance measure there by developing intelligent problem solving agent.

2.1.5.2 Steps in Problem Solving

Problem solving agent achieves success by taking following approach to problem solution -

Step 1 : Goal setting

Agent set the goal by considering the environment.

Step 2 : Goal formulation

The goals set in step 1 are formalized in the frame work. The key activity in goal formulation is

1) To observe current state. 2) To tabulate agents performance measures.

Step 3 : Problem formulation

After formulating goal, it is required to find out what will be the sequence of actions which generate goal state.

Problem formulation is a way of looking at actions and states generated because of actions, which leads to success.

Step 4 : Search in unknown environment

If the task environment is unknown then agent first tries different sequence of actions and gathers knowledge (i.e. learning). Then agent gets known set of actions which leads

to goal state. Thus agent search for describable sequence of actions this process is called as searching process.

With knowledge of environment and goal state we can design a search algorithm. A search algorithm is a procedure which takes problem as input and return its solution which represented in the form of action sequence.

Step 5 : *Execution phase*

Once the solution is given by the search algorithm then the actions suggested by the algorithm are executed. This is the execution phase. Solution guides agent for doing the actions. After executing the actions agent again formulate new goal.

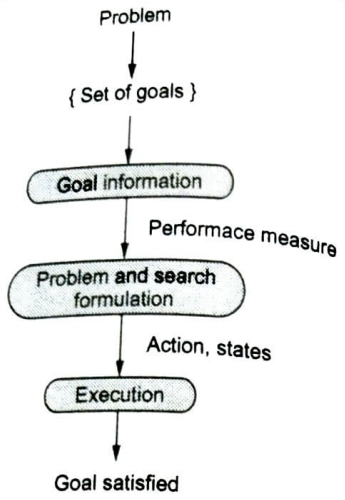


Fig. 2.1.3 Problem solving agent

2.1.3 Algorithm

Procedure or method : Problem solving agent (unknown space, percept).

Results : An action.

Input : P → percept (Environment perception)

Static :

1) A → An action sequence, initially with null value.

2) S → State - current state.

3) G → Goal - A goal initially null.

4) P → Problem - A real world situation.

State - update state (State, percept)

If (s) is empty then do

g ← Formulate goal (s)

P ← Formulate problem (s, g)

S ← Search (p)

G ← First (s)

S ← Rest (s)

Return a

Procedure

2.1.4 For Example

Consider following simple problem solving agent. Working in open-loop system. Open-loop system means agent is assumed to be working in following environment -

1) Static environment : Where in problem formulation and solution is done by ignoring the changes that can occur in environment.

2) Observable environment : Where in agent has complete knowledge of environment.

3) Discrete environment : Where in the idea of enumerating "alternative courses of actions" is implemented.

4) Deterministic environment : Where in next state is configured from current state.

- **Points to Note**

1) Above kind of working systems is called as open-loop system, because ignoring the percepts breaks loop between agent and environment.
2) In open-loop systems solutions to problem are single sequence of actions, so they cannot handle any unexpected events.
3) Also solutions are executed without paying attention to the percepts.
4) These are most easiest kind of environment to work in for agents.

2.2 State Space Search

GTU : Winter-16, 17, 19, Summer-12, 17, 18

- For finding the solution one can make use of explicit search tree that is generated by the initial state and the successor function that together define the state space. In general, we may have search graph rather than a search tree as the same state can be reached from multiple paths.

2.2.1 Construction of State Space

1) The root of search tree is a search node corresponding to initial state. In this state only we can check if goal is reached.
2) If goal is not reached we need to consider another state. Such a can be done by expanding from the current state by applying successor function which generates new state. From this we may get multiple states.
3) For each one of these, again we need to check goal test or else repeat expansion of each state.
4) The choice of which state to expand is determined by the search strategy.
5) It is possible that some state, surely, can never lead to goal state. Such a state we need not to expand. This decision is based on various conditions of the problem.

2.2.2 Terminology used in Search Trees

1) Node in a tree : It is a book keeping data structure to represent the structure configuration of a state in a search tree.

2) State : It reflects world configuration. It is mapping of state and action to another new state.

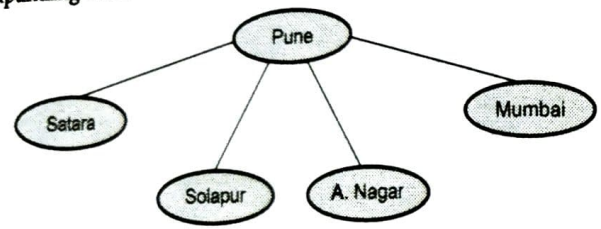
3) Fringe : It is a collection of nodes that have been generated but not yet expanded.

4) Leaf node : Each node in fringe is leaf node (as it does not have further successor node).

For example :

a) The initial state

b) After expanding Pune



c) After expanding Satara

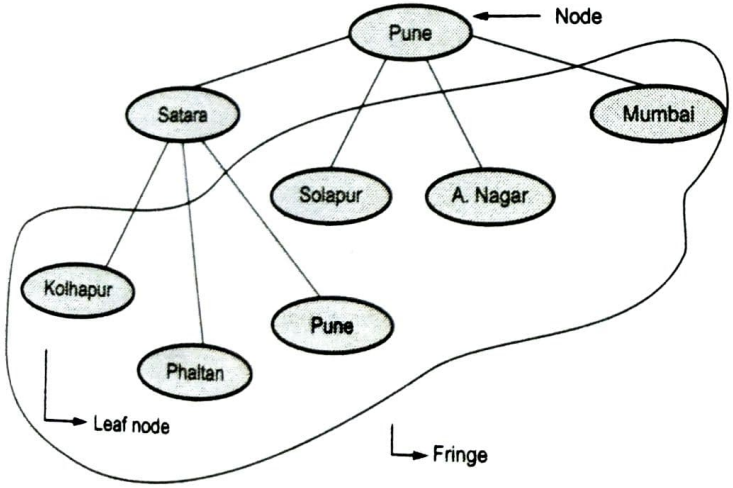


Fig. 2.2.1 Partial search trees for finding route from pune to chennai

5) Search strategy : It is a function that selects the next node to be expanded from current fringe. Strategy looks for best node for further expansion.

For finding best node each node needs to be examined. If fringe has many nodes then it would be computationally expensive.

The collection of un-expanded nodes (fringe) is implemented as queue, provided with, the sets of operations to work with queue. These operations can be CREATE Queue, INSERT in Queue, DELETE from Queue and all necessary operations which can be used for general tree-search algorithm.

2.2.3 Node Representation in a Search-Tree

Formally we can represent the node of search tree with 5 components,

1) State : The state, in the state space to which the node corresponds;

2) Parent-node : The node in the search tree that generated this node;

3) Action : The action that was applied to the parent to generate the node;

4) Path-cost : The cost, traditionally denoted by function g(n), of the path, from the initial state to the node, as indicated by the parent pointers;

5) Depth : The number of steps along the path from the initial state.

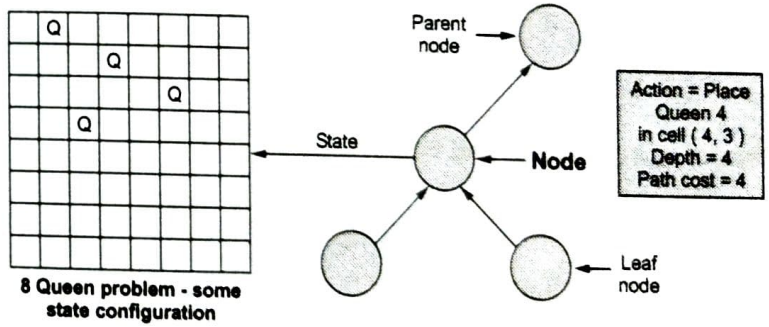


Fig. 2.2.2 Node in search tree

2.2.4 The Node Searching and Expansion Algorithms

2.2.4.1 Algorithm Tree Search

Input - Problem, fringe.

Output - Solution or failure.

1) Create initial node from problem's initial state.

2) Create fringe from initial node.

3) If fringe is empty return failure.

4) Node = first_node from_fringe.

5) If **Goal** test succeeds on node then return solution (node).

6) Expand node and add the newly generated nodes to fringe.

7) Repeat step (3) to (6).

2.2.4.2 Algorithm Expand Node

/* This algorithm will be used in Tree search for expanding the node */

Input - Node, problem

Output - A set of nodes (newly generated) (successor)

1) Initial successor set is empty.

2) For each <action, result>,
in successor function of a problem for a given state do
steps (3) to (9).

3) s = a new node.

4) STATE [s] = result.

5) Parent Node [s] = node.

6) Action [s] = action.

7) Path Cost [s] = PathCost [node] + StepCost (node, action s).

8) Depth [s] = Depth [node] + 1.

9) Add s to successor set.

10) Return successor set.

2.2.5 Measuring Problem Solving Performance

When we are solving a problem we have three possible outcomes 1) We reach at failure state 2) Solution state 3) Algorithm might get stuck in an infinite loop.

Problem solving algorithm's performance can be evaluated on the basis 4 factors -

1) Completeness : Does the algorithm surely finds a solution, if really the solution exists.

2) Optimality : Some times it happens that there are multiple solutions to a single problem. But the algorithm is expected to produce best solution among all feasible solution, which is called as optimal solution.

3) Time complexity : How much time the algorithm takes to find the solution.

4) Space complexity : How much memory is required to perform the search algorithm.

- **Major factors affecting time complexity and space complexity -**

Time and space complexity are majorly affected by the size of the state space graph, because it is the input to the algorithm. State space graph needs to be stored as well as it needs to be processed. So it is straight forward that more complex state space graph more is the space and time required.

- **The state space graphs complexity is affected by 3 factors -**

a) Branching factor : It is maximum number of successors of any node. If this value is less then less nodes will have to be search, there by getting result fast.

b) Depth of goal node : It is the depth of the shallowest goal node, where one can reach fast. If this value is small we will get goal node early.

c) Maximum length of path : It is maximum length of any path in the state space. If length value is more, complexity of state space is more. This is often measured in terms of number of nodes generated.

The major activity in searching is node expansion. The time taken for this is called as search cost. Search cost will typically depend on time complexity.

- **Path cost :** Is time bound cost which is incurred for reaching or going to a perticular node.

The total cost for algorithm is the combined cost of search cost and the path cost of the solution found along with memory usage.

Total cost = Search cost + Path cost + Memory usage.

For the problem of finding a route from Pune to Chennai, the search cost is amount of time taken by the search and the solution cost is the total length of the path.

The cost found will be helpfull for agent to find shorter paths (low cost paths).

2.3 State Space Search Strategies

GTU : Winter-12,14,17,18,19, Summer-12,13,14,16,18,19

At every stage in state space generating algorithms we need to apply searching procedure so as to reach to goal state. Search is a systematic examination at states to find path from the root state (initial state) to the goal state. The output of this procedure is the solution (goal) state.

2.3.1 Two Basic Search Strategies

1) Uninformed search (Blind search) : They have no additional information about states other than provided in the problem definition.

They can only generate successors and distinguish between goal state and non-goal state.

2) Informed search (Heuristic search) : This can decide whether one non-goal state is more promising than another non-goal state.

Key Point *All the search strategies are distinguished by the order in which the nodes are expanded.*

2.3.2 Uninformed Search Strategies

1. B.F.S.
2. D.F.S.
3. Depth limited search
4. Iterative deepening DFS
5. Bidirectional search
6. Uniform cost search

In the following section we will discuss each uninformed searching in detail. For each searching technique we will see its basic methodology, its algorithmic implementation and its performance evaluation. Performance evaluation will be done on the basis of 4 criterias we have seen previously namely,

1) Completeness 2) Optimality 3) Time complexity

4) Space complexity.

Generally it does happen that space and time complexity depends on some factors hence we will discuss them combinely.

2.3.2.1 Breadth-First Search

The procedure : In BFS root node is expanded first, then all the successor of root node are expanded and then their successor and so on. That is the nodes are expanded levelwise starting at root level.

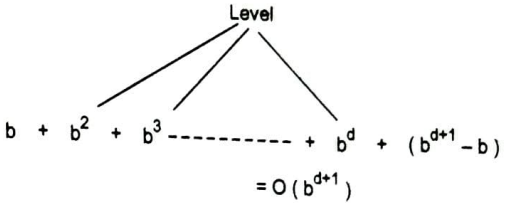
The implementation : BFS can be implemented using first, in first out queue data structure where fringe will be stored and processed. As soon as node is visited it is added to queue. All newly generated nodes are added to the end of the queue, which means that shallow nodes are expanded before deeper nodes.

The performance evaluation

1) Completeness : BFS is complete because if the shallowest goal node is at some finite depth d, BFS will eventually find it and will generate solution. (assuming that branching factor b is finite).

2) Optimality : The shallowest goal node is not necessarily optimal. BFS will yeild optimal solution only when all actions have the same cost, let it be at any depth.

3) Time and space complexity : As the level of search tree grows more time is incurred. In general if search tree is at level d then $O(b^{d+1})$ time is required, where b is the number of nodes generated from each node, starting at root node i.e. root generates b nodes and each b node generates b more and so on shown below.



Every node generated should remain in memory till its exploration. If branching factor 'b' is more then more memory will be required.

For example -

If branching factor b = 10 at some level d = 6. If we assume 10,000 nodes will be generated per second and per node 1000 bytes are required for storage.

Then total generated nodes = 10^7 which will take 19 minutes but it will take **10 Gigabytes** for storing them.

We can conclude that for BFS space complexity is major issue concerned than time complexity. Time complexity is critical issue when depth of tree increases.

Key Point *We can use uninformed methods for exponential-complexity search only when problems have smallest instances.*

Algorithm BFS (G, n)

```
//Breadth first search of G
{
    for i := 1 to n do // Mark all vertices unvisited
        visited [i] : = 0 ;
    for i := 1 to n do
        if (visited [i] = o) then BFS (i) ;
}
```

BFS

- Time complexity - $O(b^{d+1})$
- Space complexity - $O(b^{d+1})$

BFS

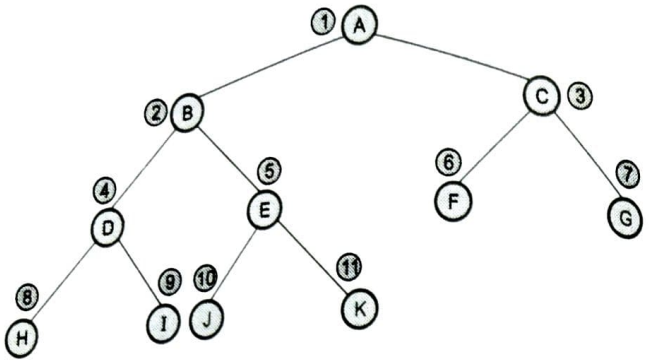


Fig. 2.3.1 Breadth-first search

D and J are goal state.

[D is found through the path] → (A-B-D) (shallowest goal)

Note : The numeric value beside each node indicates the order in which nodes are visited (reached).

2.3.2.2 Uniform Cost Search

The procedure : Uniform cost search expands the node 'n' with the lowest path cost. Uniform cost search does not care about the number of steps a path has, instead it considers their total cost. Therefore, their is a chance of getting stuck in an infinite loop if it ever expands a node that has a zero-cost action leading back to the same state.

The implementation : We can use queue data structure for storing fringe as in BFS. The major difference will be while adding the node to queue, we will give priority to the node with lowest pathcost. **(So the data structure will be a priority queue).**

The performance evaluation

1) Completeness : Uniform cost search guarantees completeness provided the cost of every step is greater than or equal to some small positive constant "c".

2) Optimality : If cost of every step is greater than or equal to some small positive constant then uniform cost search will yeild optimal solution, by reaching the goal state which has lowest path cost.

3) Time and space complexity : Uniform-cost search does not care about the number of steps a path has, but only about their total cost. Therefore, it will get stuck in an

infinite loop if it ever expands a node that has a zero-cost action leading back to the same state.

Uniform-cost search is guided by path costs rather than depths, so its complexity cannot easily be characterized in terms of b and d. Instead, let C^* be the cost of the optimal solution, and assume that every action costs at least E. Then the algorithm's worst-case time and space complexity is $O(b^{[C^*/\epsilon]})$, which can be much greater than b^d.

- Uniform cost seach

  Time complexity - $O(b^{C/\epsilon})$

  Space complexity - $O(b^{C/\epsilon})$

where - C is the cost of optimal solution.

Assuming J and D are both goal state.

A - B - E - J

Note : The costs are associated with each edge.

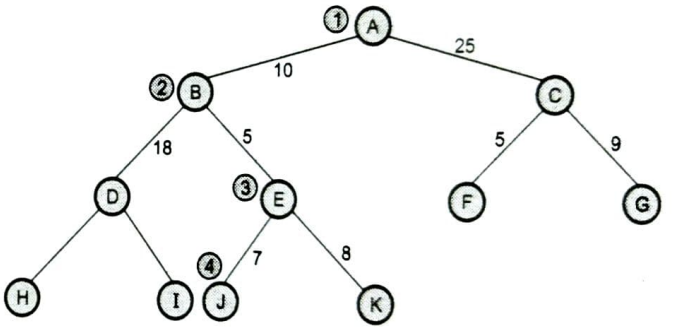


Fig. 2.3.2 Uniform cost search

Note : The numeric value beside each node indicates the order in which nodes are visited (reached).

2.3.2.3 Depth-First Search

The procedure : Depth-first search always expands the deepest node in the current unexplored node set (fringe) of the search tree. The search goes in to depth until there is no more successor node. As these nodes are expanded, they are dropped from the fringe, so then the search "backs up" to the next shallowest node (previous level node) that still has unexplored successors.

The implementation : DFS can be implemented with stack (LIFO) data structure which will explore the latest added node first, suspending exploration of all previous nodes on the path. This can be done using recursive procedure that calls itself on each of the children in turn.

The performance evaluation

1) Completeness : As DFS explores all the nodes hence it guareentees the solution.

2) Optimality : As DFS reach to deepest node first, it may ignore some shallow node which can be goal state. Therefore optimality is expected only when all states have same path cost.

3) Time and space complexity : DFS requires some moderate amount of memory as it needs to store single path from root to some node to a particular level, along with unexpanded siblings. When same node gets fully explored, its complete branch (its all descendants and itself) will get removed from memory.

With branching factor 'b' and maximum depth d, dfs requires storage of - $bd+1$ nodes.

```
Algorithm DFS (v)
//Given an undirected (OR directed) graph
//G = (V, E) with
//n vertices and an array visited initially set
// to zero, this algorithm visits all vertices
//reachable from v. G and visited [ ] are global.
{
        visited [v] : = 1;
    for each vertex w adjacent from v do
    {
        if (visited [w] = 0 then DFS (w);
    }
}
```

- Time complexity - $O(b^d)$.
- Space complexity - $O(b^d+1)$.

2.3.2.4 DFS [Depth First Search]

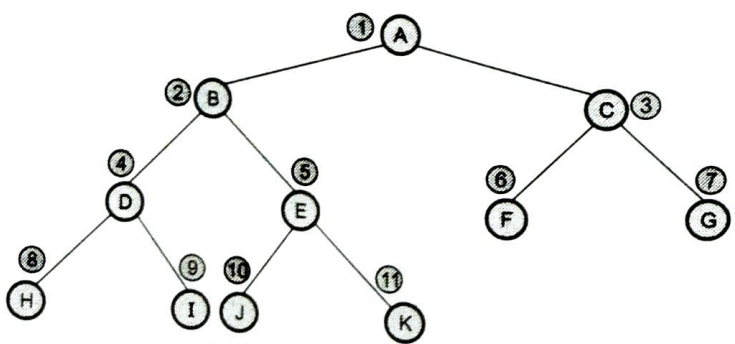


Fig. 2.3.3 Depth First Search

Goal state path - (A - B - D)

where D is a goal state.

Note : The numeric value beside each node indicates the order in which nodes are visited (reached).

A special case DFS → back tracking search : In this only one successor is generated at a time rather than all the successors and each partially expanded node remembers which successor to generate next.

This search technique uses less memory than DFS as only 'd' nodes (maximum depth) are required to store.

Drawback of DFS : DFS can make a wrong choice and get stuck going down a very long (even infinite) path when a different choice would lead to a solution near the root of the search tree.

2.3.2.5 Depth-Limited Search (DLS)

The procedure : If we can limit the depth of search tree to a certain level then searching will be more efficient. This removes the problem of unbounded trees. Such a kind of depth first search where in the depth is limited at a certain level is called as depth limited search. It solves infinite path problem.

The implementation : Same as DFS but limited to depth level *l*. DLS will terminate with two kinds of failure. The standard failure value indicates no solution and the cut-off value indicates no solution within the depth limit.

The performance evaluation

1) Completeness : DLS suffers from completeness because if we choose *l* (levels to be searched) < d (actual levels), when shallowest goal is beyond the depth limit *l*. It generally happens when 'd' is unknown.

2) Optimality : DLS is non-optimal if we choose *l* (levels to be searched) > d (actual levels) because shallowest goal state may be ignored while reaching to depth level *l*.

3) Time and space complexity : Its time **complexity is** $O(b^l)$ and space complexity is $O(bl)$. If we have knowledge of the problem, then we can decide depth limits. If we can find better depth limit then we can achieve more efficiency. **This better depth limit is termed as diameter of state space.** If problem is too complex then we are unable to find diameter of state space.

Algorithm for Depth Limited Search

```
DLS (node, goal, depth)
{
```

```
if(depth >=0)
{
    if(node ==goal)
        return node
    for each child expand(node)
        DLS (child, goal, depth-1)
}
}
```

General steps for DLS

1. Determine the node where the search should start and assign the maximum search depth.
2. Check if the current node is the goal state.
   - IF not : Do nothing
   - IF yes : Return
3. Check if the current node is within the maximum search depth.
   - IF not : Do nothing
   - IF yes : a) Expand the vertex and save all of its successors in **stack**.
     b) Call DLS recursively for all nodes of the stack and go back to step 2.

DLS [Depth-Limited Search]

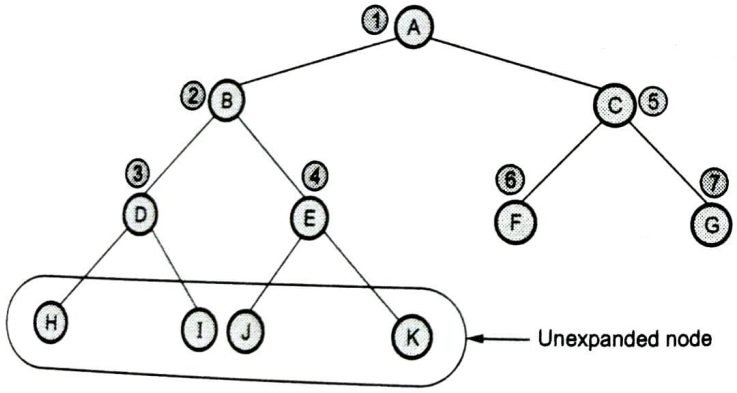


Fig. 2.3.4 Depth-Limited Search

If depth limit = 2 then nodes on level 2 are not expanded.

D found (A-B-D)

(D, E, F, G are generated but not expanded).

J though better goal than D could not reached because of algorithm technique.

Note : The numeric value beside each node indicates the order in which nodes are visited (reached).

2.3.2.6 Iterative-Deepening Depth-First Search (IDDFS)

The procedure : In iterative deepening depth-first search, dfs is applied along with the best depth limit. In each step gradually it increases the depth limit until the goal is found. It increases depth limit from level 0, then level 1, then level 2 till the shallowest goal is found at certain depth 'd'.

The implementation : Iterative deepening depth first search can be implemented similar to BFS (where queue is used for storing fringe) because it explores a complete layer of new nodes at each iteration before going on to the next layer. If we want to avoid memory requirements which are incurred in BFS then IDDFS can be implemented like uniform-cost search. The key point is to use increasing path-cost limits instead of increasing depth limits, is if we have such a implementations it is termed as iterative lengthing search.

The performance evaluation

1) Completeness : It guarantees completeness as the search does not stop until goal node is found.

2) Optimality : As iterative deepening depth first search stops when the first goal node is reached, it is not necessary that it is the optimal. (That is, the shallowest goal reached is the final. Iterative deepening depth first search will not further search for optimal solution).

3) Time and space complexity : Iterative deepening depth first search has very moderate space complex which is **O (bd)**. Its time complexity depends on branching factor (b) and the bottom most level that is depth (d). It is $O(b^d)$.

Algorithm for Iterative-Deepening Depth First Search

```
//In IDDFS algorithm we are using DLS algorithm from earlier section IDDFS (root, goal)
{
    depth = 0
    while (no solution)
    {
        solution = DLS(root, goal, depth)
        depth = depth+1
    }
    return solution
}
```

• **Example - IDDFS (Iterative Deepening Depth-First Search]**

Shallowest Goal Node D Found :

Note : The numeric value beside each node indicates the order in which nodes are visited (reached).

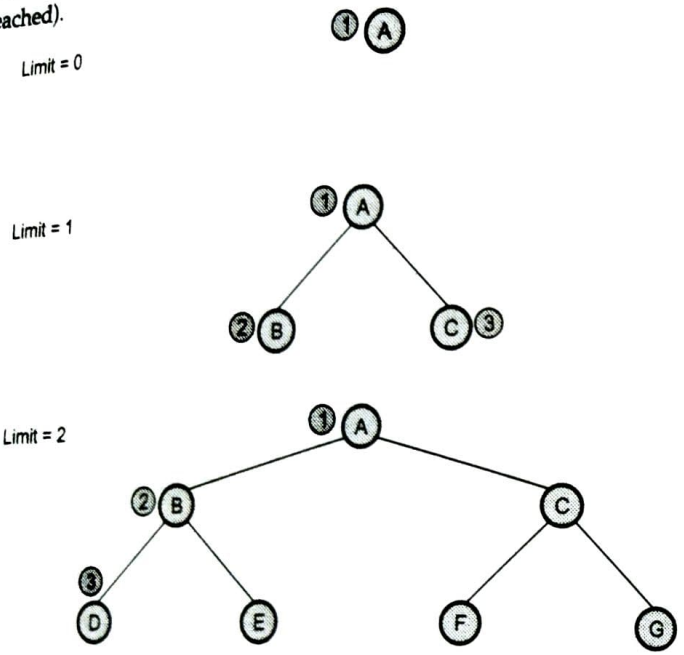


Fig. 2.3.5 Iterative Deepening Depth-First Search

2.3.2.7 Bidirectional Search

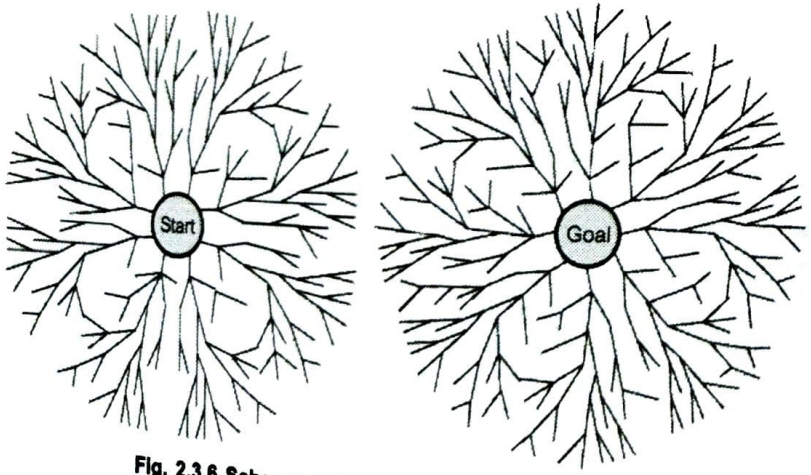


Fig. 2.3.6 Schematic view of bidirectional search

The procedure : As the name suggests bi-directional that is two directional searches are made in this searching technique. One is the forward search which starts from initial state and the other is the backward search which starts from goal state. The two searches stop when both the searches meet in the middle. [As shown in Fig. 2.3.6]

The implementation :

Bidirectional search is implemented by having one or both of the searches check, each node before it is expanded, is examined to see if it is in the fringe of the other search tree. If so solution is found. Fringe can be maintained in queue data structure, like BFS.

The performance evaluation

1) Completeness : Bidirectional search is complete if branching factor **b** is finite and both directions searches use BFS.

2) Optimality : It is optimal as the goal is being searched from both directions. So guaranteed to find optimal (best) goal state.

3) Time and space complexity : Bidirectional search has time complexity as $O(b^{d/2})$, where 'b' is branching factor. It is the important noticable point in case of backward a search because, we need to get predecessors of the node. The easiest case is when all the actions in the state space are reversible.

When one goal state is there, the backward search is very much like the forward search (like in 8 puzzle problem). If there are several explicitly listed goal states (like in part picking robot) then it needs to construct a new dummy goal state whose immediate predecessors are all then actual goal states. The worst case in bidirectional search is when the goal test gives only implicit description of some possibly large set of goal states.

For example, in the game of chess 'checkmate' is the goal state which will have many possible description.

The memory requirement for bidirectional search is also moderate which is $O(b^{d/2})$ where b = Branching factor, d = Depth, where at least one of the search tree is maintained in memory.

Bidirectional search

Space complexity - $O(b^{d/2})$

Time complexity - $O(b^{d/2})$

2.4 The Repeated States

2.4.1 Avoiding Repeated States

- When we are finding solution by searching in state space tree, some states can be repeatedly explored. This will increase the total cost. Hence we need to avoid repeated states.
- In some problems we need to explore all the repeated states again as they may reach to goal state.

For example -

- Problems where the actions are reversible, such as route-finding problems and sliding-blocks puzzles.
- Search tree for these problems are infinite. But if we cut off some repeated states We can generate only the portion of the tree that engross the state space graph.
- In some problems repeated states are not required to explore again as these states will not surely lead to goal states.
- In a state space graph we can drop multiple paths and can construct a tree where there is only one path to each state.

For example -

- Consider the 8-queen problem. Here each state can be reached through only 1 path. If we formulate the 8-queens problem such that a queen can be placed in any column, then each state with 'n' queens can be reached by n! different paths.

2.4.2 Serious Problems Caused due to Repeated States

1) Because of repeated states solvable problem can become unsolvable if the algorithm is unable to detect repeated states.
2) Because of repeated states looping paths can be generated which can lead to infinite search which is not practical approach.
3) Because of repeated states memory is unnecessaryly wasted as same state is maintained multiple times.

2.4.3 Example of State Space that Generate an Exponentially Larger Search Tree

a)

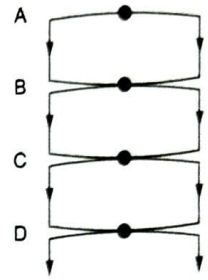


Fig. 2.4.1 State space that generate exponentially large search tree I

Fig 2.4.1 shows a state space in which there are two possible actions leading from A to B, two from B to C and so on. The state space contains d+1 states, where d is maximum depth.

b)

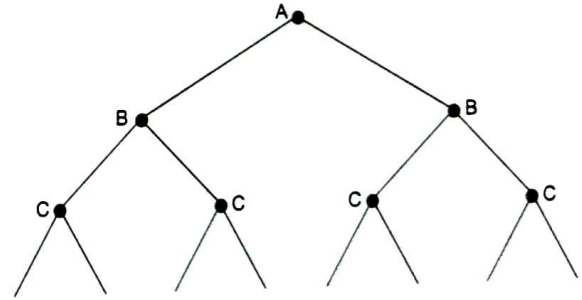


Fig. 2.4.2 State space that generate exponentially large search tree II

In the above search tree, (Fig. 2.4.2) there are 2^d branches corresponding to the 2^d paths through the space.

In the search tree shown in Fig. 2.4.3, A is a root node. On a grid each state has four successors, therefore the tree including repeated states has 4^d leaves. But as we can see, only about $2d^2$ are distinct states within d steps of any given state.

c)

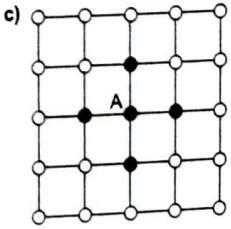


Fig. 2.4.3 State space that generate exponentially large search tree III

Note : If limit of depth in state space tree is specified then it is easy to reduce repeated states which leads to exponential reduction in search cost.

2.4.4 Algorithm that can Avoid Repeated States
(Algorithms that Forget their History are Sure to Repeate the States)

- If an algorithm remembers every state that it has visited, then it can be viewed as exploring state space graph directly.
- We can device a new algorithm called as graph-search algorithm which is more efficient than earlier tree-search algorithm. A graph-search algorithm maintains two data structures closed list and open list to avoid exporation of those node which are already explored (i.e. trying to avoid repeated states).

Close list : A closed list is a data structure maintained by algorithm which stores every expanded node. Algorithm discards the current node if it matches with node on the closed list.

Open list : A open list is a data structure maintained by algorithm which stores fringe of unexpanded node.

Graph-Search Algorithm

[Data structure required] :

- Node n
- State description
- Parent (may be backpointer) (if needed)
- Operator used to generate n (optional)
- Depth of n (optional)
- Path cost from s to n (if available)
- Open list
  - initialization : {s}
  - node insertion removal depends on specific search strategy
    - Closed list
  - initialization : { }
  - organized by backpointers to construct a solution path

[Algorithm] :

```
open : = {s} ;
closed : = { } ;
repeat
{
```



```
n : = select (open) ; /* select one node from open for expansion */
if n is a goal

then exit with success ; /* delayed goal testing */
expand (n)
/* generate all children of n put these newly generated nodes in open (check duplicates)
put n in closed (check duplicates) */
until open = { } ;
}
exit with failure
```

For the above algorithm worst case time and space requirements are proportional to the size of the state space. This may be much smaller than $O(b^d)$.

A repeated state is detected when algorithm has found two paths to same state. if the newly discovered path is shorter than the original one then it will be discarded by algorithm. Then there is a chance that graph-search could miss an optimal solution as it can discard one of the paths that can lead to optimal state.

2.5 Searching with Partial Information

All the earlier search strategies we have seen, assumed that environment is fully observable and deterministic.

In such a environment agent -

1) Knows what is the effect each action.
2) Can calculate exactly which state results from any sequence of actions.
3) Knows which state it is in.
4) Does not get new information after each action.

If environment is not fully observable and knowledge of the states and actions is incomplete then agent has different types of task environment. Such a environment leads to three distinct types of problems -

1) Sensorless problems
2) Contigency problems
3) Exploration problems.

2.5.1 Sensorless Problems (Conformant Problems)

In this type of problems -

A) Agent has no sensors at all

B) According its own knowledge agent could be in one of several possible initial states.

C) As agent has several possible initial states each action can lead to one of the several possible successor states.

- **Solving sensorless problems in a fully observable, completely known, deterministic environment ⇒**

In sensorless problem agent already have predefined knowledge about its various states. Out of these state set only one of the state is going to be initial state. Though agent has no sensor, hence no percepts, still from its own state and action it can reach to certain conclusion states, through which it can further reach to goal state. It means that we can say agent can **coerce (force to reach)** the world to reach at a perticular expected state on his own.

Steps taken to solve the problem are as follows -

1) First agent searches for **belief state** instead of physical state.

[The belief state - it is a set of states representing agent's current belief about the possible physical states it might be in. For finding such belief state agent should have reasoning for each state it can be in, based on its original knowlege].

2) Initial state is belief state which can further mapped to another belief state.

3) Action is applied to belief state by taking union of the results obtained from applying the action to each physical state in the belief state.

4) We now get a path which connects several belief states.

5) Solution is a path that leads to a belief state, all of whose members are goal states.

Note : If the physical state space has 's' states, the belief state space will have 2^s belief states.

For Example

- Black ball picker robot

2.5.2 Contigency Problem

- If environment is partially observable or if actions are uncertain, then agent can sense and perceive new information after each action. This new information can lead to new problem (or critical situation) which is called as contigency. If this situation occurs then agent needs to plan next action to handle this contigency.

• Solving contigency problem -

1) For contigency problem one needs to form a tree, here each branch of a tree may be selected depending upon the percepts received upto that point in the tree.

2) Then by searching and expanding previous level agent can reach to a goal.

In some situations for contigency problems we get purely sequential solutions.

Note -

Searching algorithm for contigency problems are more complex than algorithms we have studied so far.

The agent design is also different for these problems, because agent can act before it has found a guaranteed plan.

Agent continue to solve problem, considering new information and minimizing contigency by checking the output of its action. This is termed as interleaving of searching and execution. [That is alternately search and execution is done].

• For example :

Trading agent

• Special case of contigency problem : A contigency problem is called as adversarial problem if the uncertainty is caused by the actions of another agent. (More adverse situation).

For example -

Agent in game of chess.

2.5.3 Extreme Case of Contigency Problem

Exploration problem : When state as well as the actions of the environment are unknown the agent must act to discover them. The interleaving of searching and execution would be useful to solve exploration problem.

For example -

The boat driving robotic agent.

2.6 Production System

GTU : Winter-18,19

A **production system** is a model of computation that provides pattern-directed search control using a set of **production rules,** a **working memory,** and a **recognize-act cycle.**

- The **productions** are rules of the form **C → A,** where the LHS is known as the **condition** and the RHS is known as the **action.** These rules are interpreted as follows : *given condition, C, take action A.* The action part can be any step in the problem solving process. The condition is the pattern that determines whether the rule applies or not.
- **Working memory** contains a description of the **current state of the world** in the problem-solving process. The description is matched against the conditions of the production rules. When a conditions matches, its action is performed. Actions are designed to alter the contents of working memory.

- The **recognize-act cycle** is the control structure. The patterns contained in working memory are matched against the conditions of the production rules, which produces a subset of rules known as the **conflict set,** whose conditions match the contents of working memory. One (or more) of the rules in the conflict set is selected **(conflict resolution)** and **fired,** which means its action is performed. The process terminates when no rules match the contents of working memory.

Background

- Post (1943) proposed the production rule model as a formal theory of computation. It is equivalent in power to a **Turing machine.**
- Newell and Simon (1960s) system, **General Problem Solver** used production system as a model of **human cognition.** Production rules represent problem-solving skills stored in a person's long-term memory. The working memory represents the person's current focus of attention. Choosing rules to apply, occurs through unconscious pattern matching.
- Production systems have been developed for a wide variety of problems, ranging from algebra word problems, mathematical and logical proofs, physics problems and games.
- John Anderson (1983) proposed a production system known as ACT* as a model of human cognition and learning.
- Newell, Rosenbloom and Laird (1990) proposed a production system known as SOAR (Symbols, Operators AND Rules) as a model of human cognition and learning.
- The OPS (Official Production System) languages are based on the production system model.
- 1980s: Production systems were the basis of **rule-based expert systems.**
- Model for Human Problem Solving (SOAR, ACT*)

2.7 Issues in the Design of Search Problem

GTU : Winter-17

Design is a signature of human intelligence. It was more difficult and big challenge for AI. Designs shaped early ideas on automated problem solving and reasoning. It provided ever since, task for AI which are as follows :

i) Studying, explaining and modeling the faculties underlying intelligent behaviour.

ii) Engineering systems that are capable of exhibiting for a such behavior.

2.7.1 Issues in Engineering Design

- When basically increasing integration on one hand and distribution work on the other hand are the trends that determine the present and even more the future of

engineering design. This type of situation is by no means a paradox. When result of various engineering differ, different activities should be put together. When product is complete and launched in the market, it should be efficient and quality.

- Collaborative design transforms a chaining process into a parallel one with the aim of avoding iteration and utilizing resources, more efficiently, (Gadh-1996). Concurrent engineering has an even broader focus. Acknowledgeing that, engineering design must take into account the intrinsic requirement and properties of the process that bring to life, create, maintain and re-cycle artifacts. It embraces all main life-cycle activities such as marketing, design, manufacturing, distribution, sales, operation, maintenance, disposal and re-cycling. The expectations are greater, fewer modifications, shorter time-to market, more efficient resource utilization, better process and product quality (Sohlenius, 1992) though reliable (Alting and Legarth 1995 ; Jansen and Krause, 1996).

2.7.1.1 Design Synthesis

a) Automated search

- The search problems in design have been addressed mainly by the application of greedy numerical optimization algorithm, random re-start hill-climbing, stochastic local and global search methods. The technique is best if a design problem can be coined as a combinatorial optimization problem. They are account neither for the cost nor for the bounded resources of best for computations.
- Tangible result in economic rationally lies in the center of intelligent behavior (Doyle etal, 1996 ; Russel, 1997)

b) Model based and functional reasoning, constraint satisfaction

- Knowledge about artifacts enables reasoning about them, even if they are only partially defined. Can constrain the design process by evaluating partial design solution can be very complex problem (McGuinness and wright, 1998).
- The design process is not self automanted, rather alternatives are generated deriving the implications of the user provided data.
- The consistency of data could be preserved in the whole interactive design process. Functional models, the traditional decomposition of function to subfunction can even better direct the design process through they cannot work on multiple levels of abstraction and without human interaction (Umeda and Tomiyama, 1997).

c) Case based approach to design

- Lack of sufficient understanding of design synthesis and the resistance of expertise to formalization attempts on one hand, where the surprising complexity of automated problem solving process on the other hand led AI practitioners back to

tranditional design. Practice to the re-use of previous solution. In case based design (Maher and Garza, 1997) cases are episodic descriptions of problem together with their associated solution and trace of the solution process. Reasoning goes through the steps of adapting such stored cases that are particulary relevant to a new problem. Side-stepping the main line of reasoning, new cases be added to the case-base, CBR solve any problem to essay.

2.7.2 Issues in Design of Search Program

Following are major issues faced while designing a search program,

1. The direction in which to conduct the search that is either forward or backward reasoning is to be used.
2. How to select applicable rules for matching that would depend on data set.
3. How to represent each node of the search process. That is how the knowledge representation should be done so as to reduce the accessing time while handling the data.

2.8 AI Problem Characteristics

GTU : Summer-12,18, Winter-12,14,17

2.8.1 Problem Characteristics

To choose an appropriate method for a particular problem it is important to study the problem characteristics which are as follows :

1. **Is the problem decomposable to smaller or easier problems ?** Can the problem be broken down to smaller problems so that it can be solved independently ? Such a decomposed problem can be solved easily.

2. **Can problem solution steps be ignored or undone ?**

The problem can fall in one of the following three categories.

- **Ignorable** - In this type of problem solution steps can be ignored. For example - Theorem proving. In theorem proving if later it is known that certain step is not useful then still one can proceed further as nothing is lost due to repeated steps.
- **Recoverable** - In this type of problem solution steps can be undone and backtracked. For example - The 8-Puzzle problem. In this problem while moving from initial state to goal state one may make some wrong moves but later can backtrack and recover by making correct move.
- **Irrecoverable** - In this type of problem moves cannot be retracted. For example - Playing Chess : in the game of chess one can make wrong move but then this move is neither to be ignored nor it can be recovered.

The above knowledge about the problem helps in determining the control structure. Ignorable problems can be solved using a simple control structure that never backtracks. Recoverable problems can be solved using backtracking. Irrecoverable problems can be solved by recoverable style methods via planning.

3. **Is the problem universe predictable ?**

Further problem can be categorized as problem with certain outcome and problem with uncertain outcome. In certain outcome problems planning can be done to generate a sequence of operations that lead to a solution. For example in the 8-Puzzle problem every time move is made, it is known exactly what will happen. That is there is certain outcome !

In uncertain outcome problem planning can generate a best sequence of operations that can probably lead to problem solution. For example in Bridge game one cannot know exactly where all the cards are or what the other players will do on their turns. That is there is uncertain outcome.

For certain-outcome problems, planning can used to generate a sequence of operators that is guaranteed to lead to a solution where as for uncertain-outcome problems, a sequence of generated operators can only have a good probability of leading to a solution. Plan revision is made as the plan is carried out and the necessary feedback is provided.

4. **Is a good solution absolute or relative ?**

Another category of problem is that, when we get solution is it the best one or may be some other path can give optimal (best among all) solution. For certain problems multiple feasible solutions exist but out them one is the best and it is the required solution.

Consider following example,

1. Marcus was a man
2. Marcus was a Pompeian
3. Marcus was born in 40 A.D.
4. All men are mortal.
5. All Pompeians died when the volcano erupted in 79 A.D
6. No mortal lives longer than 150 years.
7. It is now 2008 A.D.

Question : Is Marcus alive ?

8. Marcus is mortal (1, 4)
9. Marcus'age is 1964 years. (3, 7)

10. Marcus is dead. (6, 8, 9)

——— OR ———

11. All Pompeians are died in 79 A.D.

12. Marcus is dead.

In above problem different reasoning paths lead to the answer. It does not matter which path we follow.

In Travelling Salesman Problem one has to try all paths to find the shortest one. Any-path problems can be solved using heuristics that suggest good paths to explore. For best-path problems, much more exhaustive search will be performed to get optimal solution.

5. Is the solution a state or a path ?

Consider following example,

Finding a consistent interpretation :

-"The bank president ate a dish of pasta salad with the fork".

"bank" refers to a financial situation or to a side of a river ?

"dish" or "pasta salad" was eaten ?

Does "pasta salad" contain pasta, as "dog food" does not contain "dog" ?

Which part of the sentence does "with the fork" modify ? What if "with vegetables" is there ?

In the above problem no processing record is required only final answer is outcome.

In the Water Jug Problem the path that leads to the goal must be reported.

A path-solution problem can be reformulated as a state-solution problem by describing a state as a partial path to a solution. The question is whether that is natural or not.

6. Is the problem using consistent knowledge base ? Does the problem require lots of knowledge or uses knowledge to constrain solutions ?

To solve problem correctly it is highly necessary that the knowledgebase is consistent on the scale of time and data. In some problems the knowledge base is consistent and in some it is not. For example for evaluating Boolean expression complete knowledge of theorems and laws is required which are always true. Where as in data about rate card of hourly labour would change as per time and need to be updated so that manufacturing cost is correctly computed. Many reasoning system work well in consistent domain that are un-appropriate for inconsistent domain.

7. What is the role of knowledge ?

In certain domain, knowledge plays crucial role while solving the problem. Consider following two problem,

i) Playing Chess

Here knowledge is important only to constrain the search for a solution. Additional knowledge of strategies would help to get faster solution.

ii) Reading Newspaper

Here knowledge is required even to be able to recognize a solution.

8. Does the task require periodic human-interaction with computer ?

Here one need to distinguish between 2 types of problems :

i) Solitary problem, in which there is no intermediate communication and no demand for an explanation of the reasoning process.

ii) Conversational problem, in which intermediate communication is to provide either additional assistance to the computer or additional information to the user.

Problem Classification

There is a variety of problem-solving methods, but there is no one single way of solving all problems.

Not all new problems should be considered as totally new. Solutions of similar problems can be exploited.

2.8.2 Example Problems and their Characteristics

1. Analysis of the water-Jug problem with respect to seven problem characteristics.

Consider the water Jug problem with the capacity of the two jugs of 3 and 4 litres with no marking in them. Given a water supply with a large storage using these two jugs, how one can separate 2 litres of water.

Analysis of **water jug problem** with respect to the seven problem characteristics are as follows -

i) Is the problem decomposable ?

In water jug problem, the goal is to get exactly 2 gallons of water in the 4-gallon jug. The idea of this solution is to reduce the problem into two subproblem, which is not possible. Even if one devide it into subproblem, they are not independent.

ii) Can solution be ignored or undone ?

In our problem suppose program makes a stupid move and realizes it a couple of moves later. It cannot simply play as through it had never made the stupid move. All it can do is to simply back up and start the game over from that point.

iii) Is the problem's universe predictable ?

In the problem, every time one make a move one know exactly what will happen. This means that it is possible to plan as entire sequence of move and be confident that are know what the resulting state will be. Hence problem is not universe predictable.

iv) Is a good solution absolute or relative ?

In water jug problem, the goal is to get exactly 2-gallon of water in 4-gallon jug. It does not matter which path one follows or how one got the goal state. If one do follow one path successfully to the answer, there is no reason to go back and see if some other path might also lead to a solution. Hence the good solution is absolute.

v) Is the solution a state or a path ?

Here the solution is state in which 4-gallon jug is filled with exactly 2-gallon water, but the way or path does matter.

vi) What is the role of knowledge ?

In water jug problem, the knowledge required is, just the set of rules for determining legal move and some simple control mechanism that implement an appropriate search procedure.

vii) Does the task require interaction with a person.

The problem does not require any interaction with a person. It just requires problem, set of rules and a goal state at the initial stage. Now between execution no knowledge and no interaction with person is required.

(a) State :

{(i, j), i = 0, 1, 2, 3, 4

j = 0, 1, 2, 3 }

i - Number of liters of water in 4-litre jug.

j - Number of litres of water in 3-litre jug.

b) Initial state :

{(i, j) = (0, 0) }

c) Goal state :

{ (i, j) = (2, n) }, n value can be in between 0 to 3.

d) Condition :

1) We can fill jug with pump and large storage is there of water.

2) We can pour water out of a jug to the ground.

3) We can pour water from one jug to another.

4) There is no measuring device available.

e) State space representation

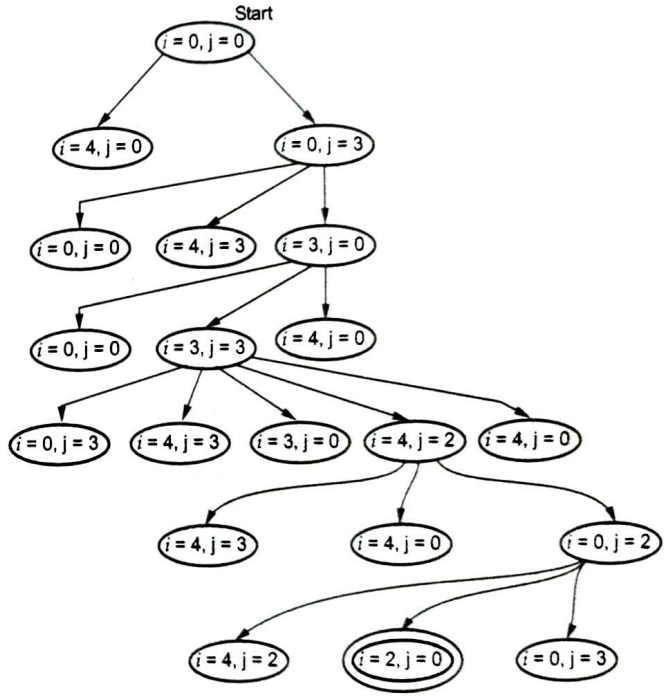


Fig. 2.8.1 State space representation

Note :

- Each state enclosed within a circle shows the resultant state after application of appropriate rule on that state.
- Initial state is represented as start state.
- Final state is enclosed within double circle.

2. Travelling salesman problem (TSP).

Analysis of **TSP** with respect to seven problem characteristics are as follows -

i) Is the problem decomposable ?

This problem is not decomposable. In this problem, the goal is to find shortest path that visit each city exactly once, and hence whole problem will be considered simultaneously.

ii) Can solution steps be ignored or undone ?

In TSP suppose program makes a stupid move means choose the path which is not a shortest path and realizes it a couple of move later. It cannot simply go further as through it had never made wrong move. All it can do is to start the game again and try to make the best of the current situation and go on from there. So solution steps cannot be ignored or undone.

iii) Is the universe predictable ?

In travelling salesman problem, every time one choose a city it is know exactly which will be the next choice, because here all the distance between each cities are known. So here it is possible to plan an entire sequence of move. One can use planning to avoid having to undo actual move. So there is nothing universe predictable.

iv) Is a good solution absolute or relative ?

In the problem, goal is to find the shortest route that visit each city exactly once. Now the next city he has to visit cannot be sure unless we also try all other path to make sure that none of them is shorter. Hence, the good solution is relative but not absolute.

v) Is the solution a state or a path ?

The goal of the problem is to find the shortest path that visit each city exactly once. For this type of problem one really must report is not the final state but the path one found to that state. Thus, a statement of a solution to this problem must be a sequence of operations that produces the final state.

vi) What is the role of knowledge ?

The required knowledge is just the knowledge of the cities to be visited, the distance between these cities and some simple control mechanism that implement an appropriate search process.

vii) Does task require interaction with a person.

In travelling salesman problem, there is no need to interact with person.

3. 8 puzzle problem

Problem characteristic	Satisfied	Reason
Is the problem decomposable ?	No	One game have Single solution
Can solution steps be ignored or undone ?	Yes	We can undo the previous move
Is the problem universe predictable ?	Yes	Problem Universe is predictable because to solve this problem it require only one person .we can predict what will be **position of blocks** in next move.
Is a good solution absolute or relative ?	absolute	Absolute solution : Once you get one solution you do need to bother about other possible solution. Relative solution : once you get one solution you have to find another possible solution to check which solution is best (i.e low cost). By considering this 8 puzzle is **absolute .**
Is the solution a state or a path ?	Path	Is the solution a state or a path to a state ? – For natural language understanding, some of the words have different interpretations. Therefore sentence may cause ambiguity. To solve the problem we need to find interpretation only, the workings are not necessary (i.e path to solution is not necessary). So In 8 puzzle winning state (goal state) describe path to state.
What is the role of knowledge ?		Lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction ?	No	Conversational· In which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or **to provide additional information to the user,** or both. In 8 puzzle additional assistance is not required.

- **State space representation for 8 puzzle problem**

In 8 puzzle problem, we have a square frame containing 8 tiles and an empty slot, i.e. total 9 sub squares. The tiles are numbered from 1 to 8. We can move the tiles in square frame by moving a tile into empty slot.

The goal state is one having the tiles in numerical order with the last square an empty slot as shown in Fig. 2.8.2.

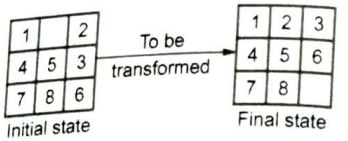


Fig. 2.8.2 State Space of 8-puzzle

Initial and final states are shown in Fig. 2.8.2 and the operators for this problem are -

UP - If the hole is not touching the top frame, move it up.

DOWN - If the hole is not touching the bottom frame, move it down.

LEFT - If the hole is not touching the left frame, move it left.

RIGHT - If the hole is not touching the right frame, move it right.

In state space representation we have all possible set of states for a given problem.

Or states space is a directed graph with all the states as nodes.

A node is said to exist if it is possible to obtain it from the initial state by application of a set of operators.

We can reach to another state by applying an operator on first state.

If the entire space representation is given of a problem, then one can easily trace the path from initial state to the goal state and identify the set of operators and their sequence necessary for reaching to goal state.

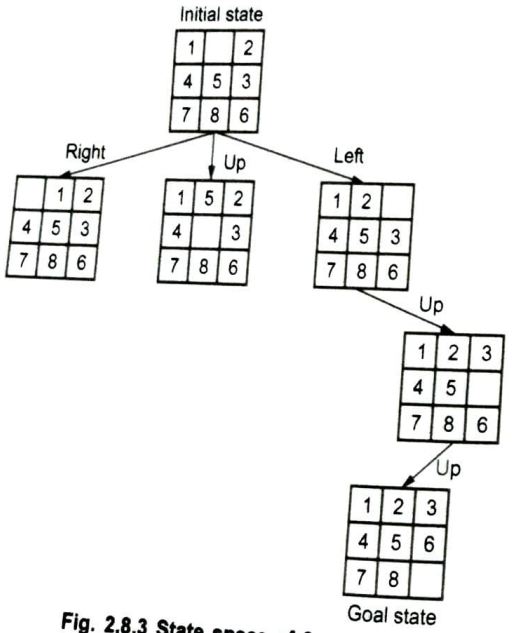


Fig. 2.8.3 State space of 8-puzzle

4. Cryptarithmatic problem

Problem characteristic	Satisfied	Reason
Is the problem decomposable ?	No	One problem have Single solution
Can solution steps be ignored or undone?	No	In actual problem we can't undo previous steps.
Is the problem universe predictable?	Yes	Problem Universe is predictable as we can figure out all possible moves.
Is a good solution absolute or relative?	Absolute	Absolute solution : once you get one solution you do need to bother about other possible solution. Relative solution : once you get one solution you have to find another possible solution to check which solution is best (i.e low cost).
Is the solution a state or a path?	Path	In this problem goal state describe path to state
What is the role of knowledge?		Lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction?	No	In cryptarithmetic additional assistance is not required

5. Chess

Problem characteristic	Satisfied	Reason
Is the problem decomposable ?	No	One game have single solution
Can solution steps be ignored or undone ?	No	In actual game (not in PC) we can't undo previous steps.
Is the problem universe predictable ?	No	Problem Universe is not predictable as we are not sure about move of other player (second player).
Is a good solution absolute or relative ?	absolute	Absolute solution : once you get one solution you do need to bother about other possible solution. Relative solution : once you get one solution you have to find another possible solution to check which solution is best (i.e low cost). By considering this **chess is absolute**.
Is the solution a state or a path ?	Path	Is the solution a state or a path to a state ? - For natural language understanding, some of the words have different interpretations. Therefore sentence may cause ambiguity. To solve the problem we need to find interpretation only, the workings are not necessary (i.e path to solution is not necessary). So in chess winning state (goal state) describe path to state.
What is the role of knowledge ?		Lot of knowledge helps to constrain the search for a solution.

Does the task require human-interaction ?	No	Conversational - In which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or **to provide additional information to the user,** or both. In chess additional assistance is not required.

6. Tower of Hanoi problem

Problem characteristic	Satisfied	Reason
Is the problem decomposable ?	No	One game have single solution
Can solution steps be ignored or undone ?	Yes	
Is the problem universe predictable ?	Yes	
Is a good solution absolute or relative ?	absolute	Absolute solution : once you get one solution you do need to bother about other possible solution. Relative solution : once you get one solution you have to find another possible solution to check which solution is best (i.e low cost). By considering this Tower of Hanoi is **absolute .**
Is the solution a state or a path ?	Path	Is the solution a state or a path to a state ? – For natural language understanding, some of the words have different interpretations . Therefore sentence may cause ambiguity. To solve the problem we need to find interpretation only , the workings are not necessary (i.e path to solution is not necessary). So In tower of Hanoi winning state(goal state) describe path to state .
What is the role of knowledge ?		Lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction ?	No	Conversational· In which there is intermediate communication between a person and the computer, either to assistance to the computer or **to provide additional information to the user,** or both. provide additional In tower of Hanoi additional assistance is not required.

State space representation

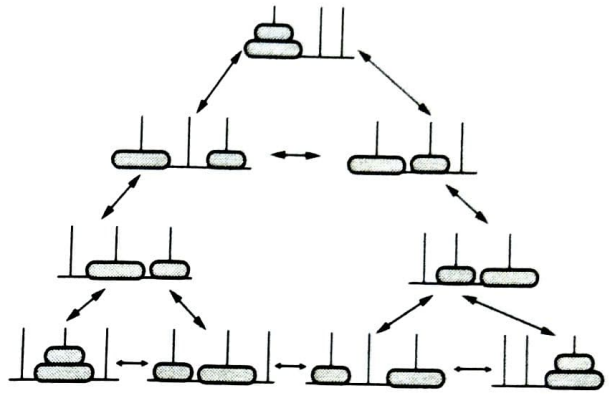


Fig. 2.8.4 Towers of Hanoi state space representation

The legal moves in this state space involve moving one ring from one pole to another, moving one ring at a time and ensuring that a larger ring is not placed on a smaller ring.

7. The missionaries and cannibals problem

- Three missionaries and three cannibals find themselves on one side of a river. They have agreed that would all like to get to the other side. But missionaries are not sure what else the cannibals have agreed to, so missionaries want to manage the trip across the river in such a way that number of missionaries on either side of the river is never less than the number of cannibals who are on the same side. The only one boat available hold only two people at a time. How can everyone get across the river without missionaries risking being eaten ?

- **Analysis of problem with respect to problem characteristics** - It encompasses a variety of specific technique, each of which is particularly effective for a small class of problem. In order to choose most appropriate method, it is necessary to analyze along seven key dimension -

1) Is the problem decomposable into a set of independent smaller or easier subproblem ?

The missionaries and cannibals problem is not decomposable because here every movement of either missionaries or cannibals effect next movement missionaries or cannibals. If one try to decompose it, he will unable to get any subproblem which is independent of others.

2) Can solution steps be ignored or at least undone if they prove ? unwise. Here many solution steps in the problem are ignored or undone when they proved unwise. undone when they proved unwise. In missionaries and cannibals problem, in initial state one have three possible move like, two missionaries move simultaneously to other side, two cannibals move simultaneously to other side, or one cannibal and one missionary can move to other side. In first case, if two missionaries move then the number of cannibals is more than the number of missionaries so this solution step can be ignored or undone. Other two possible moves can be used but second possible move (two cannibals move) will be ignored further.

3) *Is the problem universe predictable ?*

In missionaries and cannibals problem, every time one make a move, one know exactly what will happen. This means that it is possible to plan an entire sequence of move and be confident that we know what the resulting state will be. So, by explaining all the above, it means to say that, here nothing is universe predictable, means result of every possible move is exactly known.

4) *Is a good solution to the problem obvious without comparison to all other possible solution ?*

In the problem there is one and only one good solution to the problem. Hence no need to compare it with other. If there exists more than one path of the solution then one cannot be sure unless we also try all other path to make sure that none of them is more efficient.

5) *Is the desired solution a state of the world or a path to a state ?*

In missionaries and cannibals problem it is not sufficient to report that one has solved the problem and that the final state is reached. Here what we really must report is not the final state but the path that leads to the state. Thus, a statement of a solution to this problem must be a sequence of operations, that produce the final state.

6) Is a large amount of knowledge absolutely required to solve the problem, or is knowledge important only to constraint the search ?

Here there is no need of large amount of knowledge. The required knowledge is very little - just the rule for determine the legal moves and some simple control mechanism that implements an appropriate search procedure. Additional knowledge about such things as good strategy and tactics could of course help considerably to constraint the search and speed up the execution of the program.

7) Can a computer, given the problem return the solution, or will the solution of the problem require interaction between the computer and person ?

In the problem, there is no need of interaction between the computer and person. Here, given the problem, set of rule and a final state it returns the solution with each intermediate step.

- **State space representation of missionaries and cannibals problem -** In state space representation, every missionary is denoted by M and cannibals is denoted by C. Numbers of M's and C's show the number of missionaries and cannibals. Circle represents the state and arrow represents the forward and backward move.

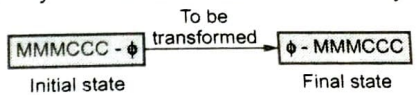


Fig. 2.8.5 Initial and final state of the problem

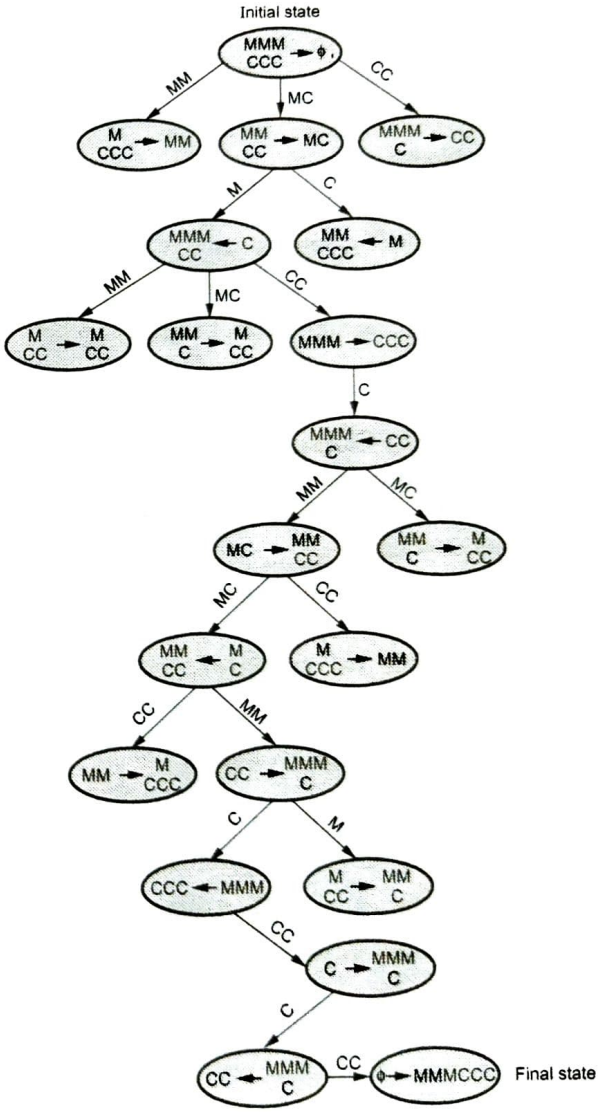


Fig. 2.8.6 State space representation

2.9 Additional Problems

GTU : Winter-12,14,15,18,19, Summer-12,16,17,19

- We have seen problem formulation. As we can see there are variety of task environments ranging from easier to complex ones, therefore our problem solving approach should be realistic, which can give solution for any type of task environment, which inturn is useful for mankind.
- Basically we can distinguish problems in two catagories. Toy problems and real-world problems.

2.9.1 Toy Problem

A Toy problem is a problem which illustrates various problem-solving methods.

Characteristics of Toy problem

1) For the Toy problem exact and precised description can be given.
2) These problem provide basis for solving some real-life problems.
3) They can be used by researchers to compare performance of algorithms.

For example : 8-queen puzzle, vaccum world, ball picker robot.

2.9.2 Real World Problem

- A real-world problem is a problem which needs to be solved so that its solution can be utilized in practical life. They will not have some predefined. Well described, single specification. Infact while considering these problem general formulation needs to be done.
- People do care about the solutions of real-world problem as they are benefited from it.

For example : Route finding for a trip, Travelling salesman problem, Robot navigation, Car reversing guide.

2.9.3 Problem Formulation for Toy Problems

Now let us see some toy problem examples.

Each example gives all 4 aspects of problem formulation namely

1) Initial state 2) Successor function 3) Goal state 4) Path cost.

Also we list all the possible states of problem solution, through which agent can pass or move.

Example 1 : Ball Picker Robot Problem Formulation

Problem statement : 2 buckets are full of balls. Either a ball is white colour or red colour. Objective is to pick all black colour balls from bucket.

1) Initial state : Any state can be designated as initial state.

2) Successor function : This function generates legal states that result from trying the three actions (Left, Right, **Pick Black** colour ball).

3) Goal test : This checks if all **Black** coloured balls from both the buckets are picked up.

4) Path cost : Each step cost is 1, so path cost is the number of steps in the path.

5) States : The agent is at one of the 2 location (at bucket A or at bucket B), from each of which black colour balls may have been picked up completely or may not be.

Thus there are 2 (locations) $\times$ 2^{2} (Two possibilities) = 8 possible world states.

- **Comparison with real world -**

If we compare with real world situation we can see that above toy problem has all well specified description. Agent has discrete location (in real world many locations can be there, also as time passes over, more locations can come up). Agent has discrete pick up item i.e. **Black** colour ball (In real world this items can have more dimensions). Also once all **Black** colour balls are picked up from bucket then again new black colour balls are not expected. (Where as in real world we may expect bucket to have new **Black** colour balls).

- **The state space representation for ball picker robot**

(See Fig. 2.9.1 on next page)

Example 2 : 8-Queen Problem

Problem Statement : Given a chess board of 8×8 size, objective is to place 8 queens on a chess board such that no two queens are attacking. (A queen attacks any queen in the same row, column or diagonal).

1) Initial state : No queens on the board.

2) Successor function : Add a queen to any empty square, such that it do not attack other queen.

3) Goal test : 8-queens on the board such that no queens attack each other.

4) Path cost : Each step costs 1, so the path cost is the number of steps in the path.

5) States : Any arrangements of 1 to 8 queens on the board is a state.

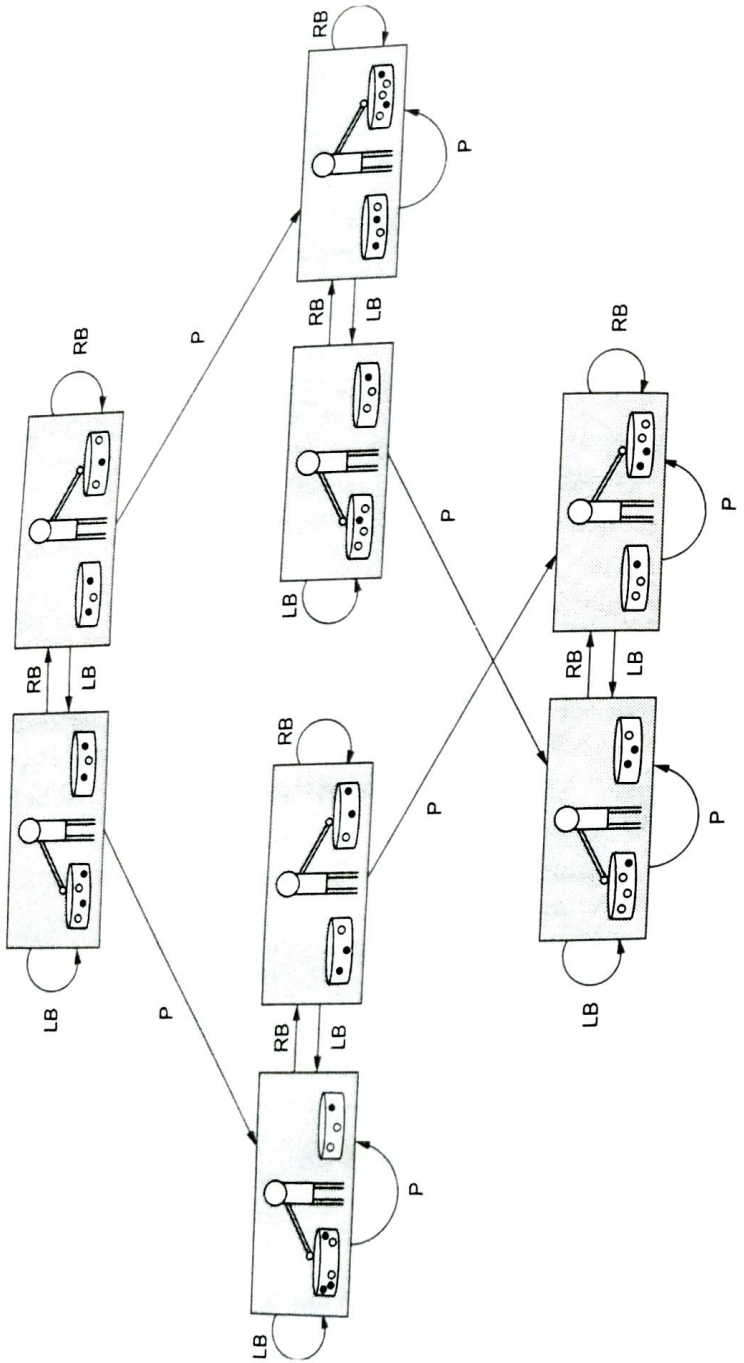
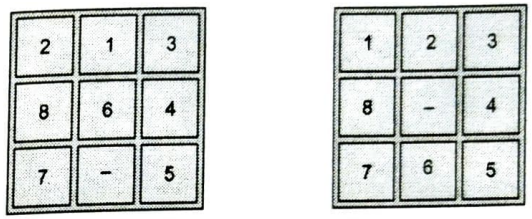


Fig. 2.9.1 The state space representation for ball picker robot

This formulation has, $64 \times 63 \times ... 57 \approx 3 \times 10^{14}$ possible sequences to investigate. But the condition of non-attacking reduces the states to 2057 from 3×10^{14}.

Example 3 : 8-Puzzle

Problem Statement : Consists of 3×3 board with eight numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The object is to reach a specified goal state, such as the one shown on the right of the figure.



(a) Initial state **(b) Goal state**

Fig. 2.9.2 8 Puzzle problem

1) States : A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.

2) Initial state : Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states.

3) Successor function : This function generates the legal states that result from trying the four actions. (blank moves, left, right, up or down).

Example 4 - Cryptarithmetic

Problem Statement : Find an assignment of digits (0, ..., 9) to letters so that a given arithmetic expression is true.

For example - Applying digit to a letter.

	F	O	R	T	Y	→		2	9	7	8	6
+			T	E	N	→	+			8	5	0
+			T	E	N	→	+			8	5	0
S	I	X	T	Y	→			3	1	4	8	6

1) States : Puzzle with letters and digits.

2) Initial state : Only letters present.

3) Successor function (operators) : Replace all occurrences of a letter by a digit not used yet.

4) Goal test : Only digits in the puzzle. Calculation is correct.

Solution for above example -

F = 2, O = 9,
R = 7, T = 8,
Y = 6, E = 5,
N = 0, I = 1,
X = 4, S = 3.

5) Goal test : This checks whether the state matches the goal configuration.

6) Path cost : Each step costs 1, so the path cost is the number of steps in the path.

Example 5 - Missionaries and Cannibals

- **Problem Statement :** There are 3 missionaries, 3 cannibals, and 1 boat that can carry upto two people on one side of a river.

Goal : Move all missionaries and cannibals across the river.

Constraint : Missionaries can never be out numbered by cannibals or either side of the river, or else the missionaries are killed.

1) States

- Number of missionaries, cannibals, and boats on the banks of a river.
- Illegal states : Missionaries are outnumbered by cannibals on either bank.

2) Initial states

- All missionaries, cannibals and boats are on one bank.

3) Successor function (operators)

- Transport a set of upto two participants to the other bank

{1 missionary} or {1 cannibal } or {2 missionaries } or {2 cannibals } or {1 missionary and 1 cannibal }

4) Goal test

- Nobody left on the initial river bank.

5) Path cost

- Number of crossings.
- Also known as "Goats and cabbage, Wolves and sheep", etc.

Example 6 - Water Jug Problem

Problem Statement : Given 3 jugs (9, 5 and 3 liters), a water pump and a sink, how do you get exactly 7 liters into the 9 litre jug.

1) State : (X, Y, Z) for liters in jugs 1, 2 and 3. Integers 0 to 9 are assigned to all possible permutations of 1, 2, 3.

Operations -

Empty jug, fill jug

Ex = fill (0, 5, 0)

2) Initial state : (0, 0, 0)

3) Goal state : (X, X, 7)

4) Solution sequence : (5, 0, 0 (0, 5, 0 (0, 0, 0 etc)

2.9.4 Real-World Problem Examples

Example 1 - General Route Finding Problem

Problem Statement : 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 air line travel planning systems.

1) States : Locations

2) Initial state : Starting point

3) Successor function (operators) : Move from one location to another.

4) Goal test : Arrive at a certain location.

5) Path cost : May be quite complex, which can involve factors like,

Money, time, travel, comfort, scenery, ... etc.

- **Simplified Example of Route Finding Problem [Airline Travelling Problem]**

Problem Statement : Starting from initial location one has to reach at the specified destination by some prespecified time.

1) States : Each state is represented by a location (e.g. an airport) and the current time.

2) Initial state : This is specified by the problem.

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

4) Goal test : Are we at the destination by some prespecified time ?

5) Path cost : This depends on monetary cost, waiting time, flight time, customs and immignation procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on.

Example 2 - Travelling Sales Person Problem

Problem Statement : It is a touring problem in which each city must be visited exactly once. The aim is to find the shortest tour. The problem is known to be NP-hard.

1) States

- Locations/cities.
- Illegal states.

a) Each city may be visited only once.

b) Visited cities must be kept as state information.

2) Initial state

- Starting point.
- No cities visited.

3) Successor function (operators)

- Move from one location to another one.

4) Goal test

- All locations visited.
- Agent at the initial location.

5) Path cost

- Distance between locations.

In addition to planning trips for traveling sales persons, these algorithms have been used for tasks such as planning movements of automatic circuit-board drills, and for stocking machines on shop floors.

Example 3 - VLSI Layout Problem

Problem Statement : A VLSI 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.

1) States

- Positions of components, wires on a chip.

2) Initial state

- Incremental : No component placed.
- Complete-state : All components place (e.g. randomly, manually).

3) Successor function (operators)

- Incremental : Place components, route wire.
- Complete-state : Move component, move wire.

4) Goal test

- All components placed.
- Components connected as specified.

5) Path cost

- May be complex.

One can consider distance, capacity, number of connections per component, for path cost computation.

• Detail discription of VLSI layout problem

When logical design is made, the layout problem come and the problem is divided into two parts 1) Cell layout 2) Channel routing as below -

1) Cell layout

a) The basic components of circuit are grouped into cells, each cells perform some recognition. The cells of standard size are connected to each of the other cell.

b) The overlapping between the cell component is avoided. Some sort of room is provided for connecting wires.

2) Channel routing

a) The cell components are fixed on a chip.

b) The channel routing task is to search a perticular route for each wire. The functional task is achieved through the gaps between the cells.

c) The complex algorithm are designed to perform the channel routing. The degree of complexity is so high, but it is always possible to solve problem with specific algorithm.

Example 4 - Robot Navigation

Problem Statement : Robot navigation is a generalization of the route-finding problem. Rather than a discrete set of routers, a robot can move in a continuous space with (in

principle) an infinite set of possible actions and states. For a circular robot moving on a flat surface the space is essentially two-dimensional. When the robot has arms and legs or wheels then it must also be controlled, the search space becomes many-dimensional.

1) States

- Locations.
- Position of actuators.

2) Initial state

- Start position (dependent on the task).

3) Successor function (operators)

- Movement, actions of actuators.

4) Goal test

- Task-dependent.

5) Path cost

- May be very complex.
- Distance, energy consumption.

Example 5 - Assembly Sequencing

Problem Statement : In assembly problems, the aim is to find an order in which to assemble the parts of some object. If the wrong order is choosen, there will be no way to add some part later in the sequence without undoing some of the work already done. Checking a step in the sequence for feasibility, is a difficult geometrical search problem, closely related to robot navigation.

1) States

- Location of components.

2) Initial state

- No components assembled.

3) Successor function (operators)

- Place component.

4) Goal test

- System fully assembled.

5) Path cost

Number of moves

Another example of assembly sequencing is protein design, in which the goal is to find a sequence of amino acids that with fold into a three-dimensional protein with the right properties to cure some disease.

Example 6 - Monkey - Banana Problem

Problem Statement : A monkey is in a room containing a box and a bunch of bananas. The bananas are hanging from the ceiling out of reach of the monkey. What sequence of actions will allow the monkey to get the bananas ? (The monkey is supposed to go to the box, puch it under the bananas, climb on top of it and grasp the bananas).

Goal Monkey to grasp all the bananas in hand.

1) States - the states can be, monkey is on the floor, monkey is on the box, monkey is under the bananas, the position of box, monkey has bananas.

2) Initial states - Monkey on the floor without bananas -

3) Successor function (operators) -

- Walk from (Monkey, Box)
- Push (Box)
- Climb (Monkey)
- Grasp (Monkey, Bananas)

4) Goal test

- Grasp (Monkey, Bananas, Yes)

5) Path cost

- The number of moves taken by monkey to get bananas.

Answer in Brief

1. *How will you measure the problem solving performance ?* **(Refer section 2.2)**
2. *What is application of BFS ?* **(Refer section 2.3)**
3. *State on which basis search algorithms are chosen ?* **(Refer section 2.1)**
4. *Evaluate performance of problem solving method based on DFS algorithm.* **(Refer section 2.3)**
5. *How a problem is formally defined ? List down the components of it.* **(Refer section 2.1)**
6. *Formulate the 8-puzzle problem using standard formulation and 8-queen problem using incremental and complete state formulation.* **(Refer section 2.8)**
7. *What do you mean by state space search ?* **(Refer section 2.2)**
8. *Discuss any 2 uniformed search methods with examples.* **(Refer section 2.3)**
9. *Write algorithm for BFS and DFS.* **(Refer section 2.3)**
10. *Write algorithm for BFS, when does one prefer it ? Also discuss A* algorithm.* **(Refer section 2.3)**