2
Problem Solving Through AI

2.1 INTRODUCTION

In the last chapter, we have tried to explain that Artificial Intelligence is the science and technology applied to make machines intelligent. The ultimate aim of researchers is to develop universal intelligent system to match the intelligence capabilities of human beings. In this regard, lot of progress has been made and considerable amount of success has been achieved, although, universal intelligent system is still a dream. Scientists have developed techniques to use AI in a limited domain and have successfully developed many AI systems, which work in a problem specific domain and show expertise not only matching those of human experts, but also exceeding those in many ways and in many applications. It should be kept in mind that for the time being, the most important application of AI is to develop intelligent systems to solve real world problems, which otherwise take considerable amount of time and human efforts, and hence, become uneconomical as well as inefficient at times. Hence, problem solving becomes major area of study, which involves methods and various techniques used in problem solving using AI.

In this chapter, we will discuss the methods of solving real world problems using Artificial Intelligence (AI) techniques. In real world, there are different types of problems. We will try to explain various characteristics of problems and their solution methodologies.

The method of solving problem through AI involves the process of defining the search space, deciding start and goal states and then finding the path from start state to goal state through search space. The movement from start state to goal state is guided by set of rules specifically designed for that particular problem (sometimes called production rules). The production rules are nothing but valid moves described by the problems.

Let us first discuss some terms related with AI problem solution methodology:

Problem	It is the question which is to be solved. For solving a problem it needs to be precisely defined. The definition means, defining the start state, goal state, other valid states and transitions.

techniques are devised for this phase. These KR methodologies have their respective advantages and disadvantages, and specific techniques are suitable for specific applications. Various KR techniques are described in detail in following chapters.

2.1.4 Finding the Solution

After representation of the problem and related knowledge in the suitable format, the appropriate methodology is chosen which uses the knowledge and transforms the start state to goal state. The techniques of finding the solution are called search techniques. Various search techniques are developed for this purpose. They are dealt-with in detail in one of the following sections and also in next chapter.

2.2 REPRESENTATION OF AI PROBLEMS

In this section, we would learn the technical aspects of problem representation required from computational perspective. From this orientation, the representation of AI problems can be covered in following four parts:

1. *A lexical part*: that determines which symbols are allowed in the representations of the problem. Like the normal meaning of the lexicon, this part abstracts all fundamental features of the problem.
2. *A structural part*: that describes constraints on how the symbols can be arranged. This corresponds to finding out possibilities required for joining these symbols and generating higher structural unit.
3. *A procedural part*: that specifies access procedures that enable to create descriptions, to modify them, and to answer questions using them.
4. *A semantic part*: that establishes a way of associating meaning with the descriptions.

Let us understand these steps by an example presented below:

We are given the problem of "playing chess". To define the problem, we should specify the starting position of chess board, the rules that define the legal moves and the board positions that represent a win for one side or the others. Here, the:

1. Lexical part contains the board position, which may be an 8 x 8 array where each position contains a symbol indicating an appropriate chess piece in the official chess opening position. The goal is any board position in which opponent does not have a legal move and his king in under attack.
2. The structural part describes the legal moves. The legal moves provide the way of getting from an initial state to a goal state. They are described as set of rules consisting of a left hand side that serves as a pattern to be matched against the current board position and a right hand side that describes changes to be made to board position to reflect the move.
3. The procedural part will be methods for applying appropriate rule for winning the game. It will include representing the "set of winning moves"

of standard chess playing. Out of all legal moves, only those moves, which bring the board position to a winning position of respective player are captured and stored in procedural part.

However, in chess the total '*legal moves*' are of the order of 10^{120}. Such a large number of moves are difficult to be written, so only '*useful moves*' are written. It should be noted that there is a difference between legal moves and useful moves. The legal moves are all those moves which are permissible according to game rules and useful moves will be those legal moves which bring the game in a winning position. Hence the 'useful moves' will be a subset of 'legal moves'. The state space is total valid states possible in a problem and finding a problem solution is "starting at an initial state, using a set of rules to move from one state to another and attempting to end up in one of a set of final states."

4. The semantic part is not required in "chess game" because, there is no hidden meaning associated with any piece and all the move meanings are explicit. However in natural language processing type of applications, where there is "meaning" associated with words and "complete message" associated with sentence, the semantic part captures and stores, the meaning of words and message conveyed by sentence.

Thus, in AI, to design a program for solution, the first step is the creation of a formal and manipulable description of problem itself. This includes performing following activities:

1. Define a state space that contains all the possible configurations of the relevant objects.
2. Specify one or more states within that space, which describe possible situations from which problem solving process may start. These are called initial states.
3. Specify one or more states that would be acceptable as solutions to the problem. These are called goal states.
4. Specify a set of rules that describes the action (operators) available.

2.3 PRODUCTION SYSTEM

In the above section, we have discussed basic aspects of AI problem solution. An AI system developed for solution of any problem is called production system. Once the problem is defined, analyzed and represented in a suitable formalism, the production system is used for application of rules and obtaining the solution. A production system consists of following components:

1. *A set of production rules*, which are of the form $P \rightarrow Q$. Each rule consists of a left hand side constituent that represents the current problem state and a right hand side that represents a result or generated output state. A rule is applicable if its left hand side matches with the current problem state. Thus, the left side of the rule determines the applicability of the rule and the right side that describes the output, if rule is applied. It is important to note

4. *Knowledge intensive*: The knowledge base of production system stores extensive and pure knowledge. This part contains no control or programming information. The problem of semantics is resolved by representing the knowledge in proper structure.
5. *Opacity*: Along with the advantages, there are certain disadvantages also associated with production systems. Opacity is the problem generated by combination of production rules. Though, the individual production rules may be models of clarity, the combined operation and effects of control program may be opaque. The opacity is generated because of less prioritization of rules.
6. *Inefficiency*: Sometimes, several of the rules become active during execution. A well devised control strategy reduces this problem. As the rules of production system are large in number and they are hardly written in hierarchical manner, it requires exhaustive search through all the production rules for each cycle of control program. It makes the functioning inefficient.
7. *Absence of learning*: The simple rule based production system does not store the results of computations for later use. Hence, it does not exhibit any type of learning capabilities.
8. *Conflict Resolution*: The rules in ideal production system should not have any type of conflict. The new rule whenever added in the database should ensure that it does not have any conflict with the existing rules. Besides this, there may be more than one number of rules that can be applied (typically called *fired*) in one situation. While choosing the rule also, the conflict should not happen. If conflict is found, it should be resolved in following ways:
   (i) Assign priority to the rules and fire the rule with the highest priority. This method is used in many expert systems. Its virtue lies in its simplicity, and by ordering the rules in the approximate order of their firing frequency. This can be made using a relatively efficient strategy.
   (ii) Use a longest matching strategy. This means that fire a rule having largest number of matching constraints. Its advantage is that the discrimination power of a strict condition is greater than of a more general condition. A rule with more constraints provides more knowledge.
   (iii) Choose most recently used rule for firing. Its advantage is that it represents a depth first search, which follows the path of greatest activity.

2.3.2 Characteristics of Production System

Now let us describe some of the main characteristics of production system from the aspect of their storage in computer system. The production systems can be of various types, but the most popular production system is monotonic system. A monotonic production system is a production system in which the application of one rule never prevents the later application of another rule. The characteristics of production systems are presented below:

Board position: = {1,2,3,4,5,6,7,8,9}

An element contains the value 0, if the corresponding square is blank; 1, if it is filled with "O" and 2, if it is filled with "X".

Hence starting state is {0,0,0,0,0,0,0,0,0}

The goal state or winning combination will be board position having "O" or "X" separately in the combination of ({1,2,3}, {4,5,6}, {7,8,9},{1,4,7},{2,5,8}, {3,6,9}, {1,5,9}, { 3,5,7}) element values. Hence two goal states can be {2,0,1,1,2,0,0,0,2} and {2,2,2,0,1,0,1,0,0}. These values correspond to the goal states shown in the figure.

The start and goal state are shown in Fig. 2.2.

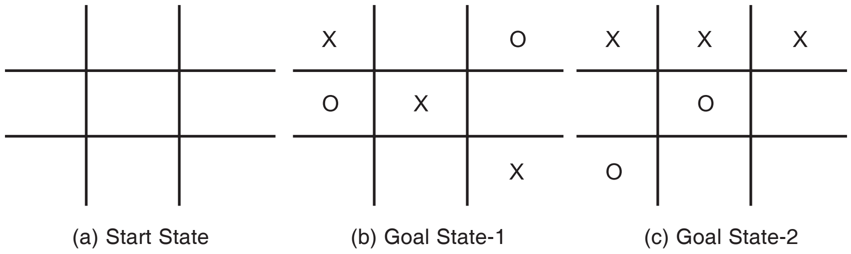


Fig 2.2: Start and goal states of TIC-TAC-TOE

Any board position satisfying this condition would be declared as win for corresponding player. The valid transitions of this problem are simply putting '1' or '2' in any of the element position containing 0.

In practice, all the valid moves are defined and stored. While selecting a move it is taken from this store. In this game, valid transition table will be a vector (having 3^9 entries), having 9 elements in each.

2.5.2 Water-Jug Problem

This problem is defined as:

"We are given two water jugs having no measuring marks on these. The capacities of jugs are 3 liter and 4 liter. It is required to fill the bigger jug with exactly 2 liter of water. The water can be filled in a jug from a tap".

In this problem, the start state is that both jugs are empty and the final state is that 4-liter jug has exactly 2 liters of water. The production rules involve filling a jug with some amount of water, filing the water from one jug to other or emptying the jug. The search will be finding the sequence of production rules which transform the initial state to final state.

Part of search tree of water jug problem is shown in Fig. 2.3.

sequence to transform the start state to goal state. One solution is applying the rules in the sequence 2, 9, 2, 7, 5, 9. The solution is presented in the following Table 2.1.

Table 2.1: Production rules applied in Water-Jug problem

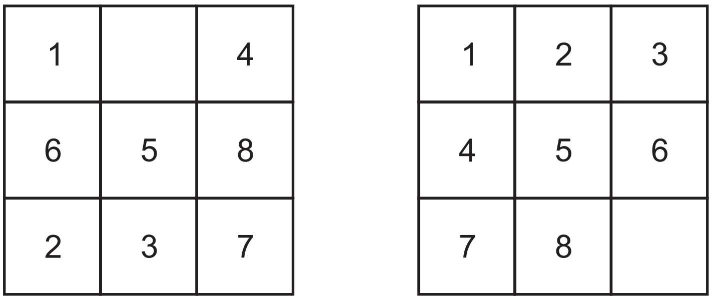
Rule applied	Water in 4-liter jug	Water in 3-liter jug
Start state	0	0
2	0	3
9	3	0
2	3	3
7	4	2
5	0	2
9	2	0

2.5.3 8-Puzzle Problem

The 8-puzzle problem belongs to the category of “sliding-block puzzle” types of problems. It is described as follows:

> *“It has set of a 3x3 board having 9 block spaces out of which, 8 blocks are having tiles bearing number from 1 to 8. One space is left blank. The tile adjacent to blank space can move into it. We have to arrange the tiles in a sequence.”*

The start state is any situation of tiles, and goal state is tiles arranged in a specific sequence. Solution of this problem is reporting of “movement of tiles” in order to reach the goal state. The transition function or legal move is any one tile movement by one space in any direction (i.e. towards left or right or up or down) if that space is blank. It is shown in following Fig. 2.4:



(a) Start state (b) Goal state

Fig. 2.4: Start and Goal states of 8 puzzle problem

Here the data structure to represent the states can be 9-element vector indicating the tiles in each board position. Hence, a starting state corresponding to above configuration will be {1, blank, 4, 6, 5, 8, 2, 3, 7} (there can be various different start positions). The goal state is {1, 2, 3, 4, 5, 6, 7, 8, blank}. Here, the possible movement outcomes after applying a move can be many. They are represented as tree. This tree is called state space tree. The depth of the tree will depend upon the number of steps in the solution. The part of state space tree of 8-puzzle is shown in Fig 2.5.

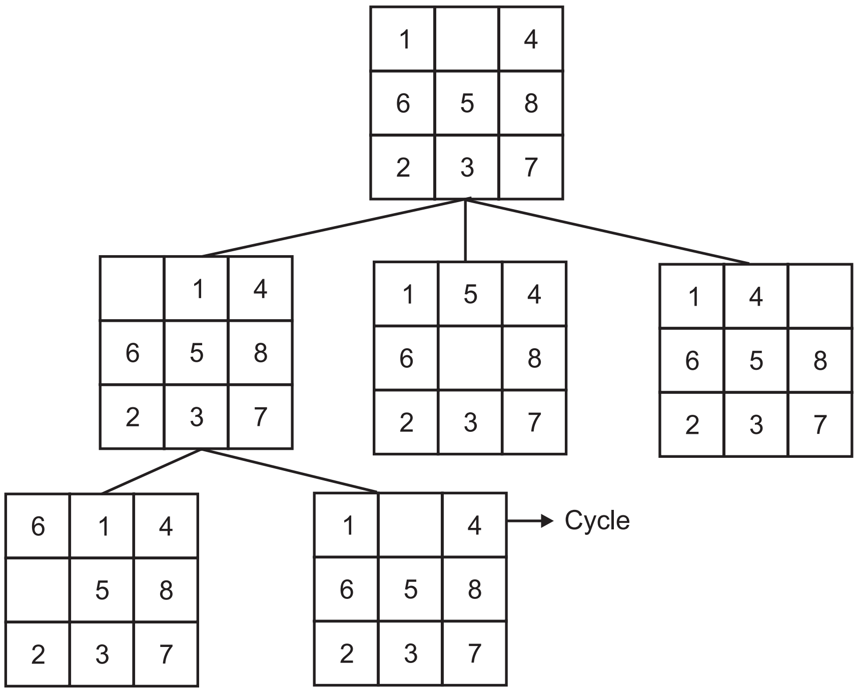


Fig 2.5: Partial search tree of 8-puzzle problem

2.5.4 8-Queens Problem

This problem is presented as follows:

"We have 8 queens and a 8 x 8 chessboard having alternate black and white squares. The queens are placed on the chessboard. Any queen can attack any another queen placed on same row, or column, or diagonal. We have to find the proper placement of queens on the chessboard in such a way that no queen attacks other queen".

The 8-queen problem is shown in following diagram:

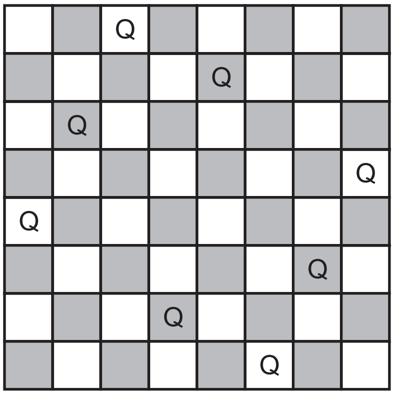


Fig 2.6: A possible board configuration of 8 queens problem

2.5.5 Chess Problem

It is a normal chess game. In a chess game problem, the start state is the initial configuration of chessboard. The final or goal state is any board configuration, which is a winning position for any player (clearly, there may be multiple final positions and each board configuration can be thought of as representing a state of the game). Whenever any player moves any piece, it leads to different state of game.

It is estimated that the chess game has more than 10^{120} possible states. The game playing would mean finding (or searching) a sequence of valid moves which bring the board from start state to any of the possible final states.

The start state of chess game is shown in Fig 2.7.

2.5.6 Missionaries and Cannibals Problem

The problem is stated as follows:

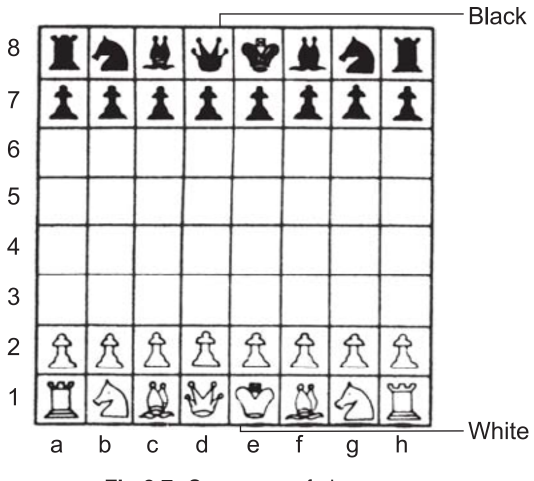


Fig 2.7: Start state of chess game

"Three missionaries and three cannibals are present at one side of a river and need to cross the river. There is only one boat available. At any point of time, the number of cannibals should not outnumber the number of missionaries at that bank. It is also known that only two persons can occupy the boat available at a time."

The objective of the solution is to find the sequence of their transfer from one bank of river to other using the boat sailing through the river satisfying these constraints.

We can form various production rules as presented in water-jug problem. Let Missionary is denoted by 'M' and Cannibal, by 'C'. These rules are described below:

Rule 1 :	(0, M)	:	One missionary sailing the boat from bank-1 to bank-2
Rule 2 :	(M, 0)	:	One missionary sailing the boat from bank-2 to bank-1
Rule 3 :	(M, M)	:	Two missionaries sailing the boat from bank-1 to bank-2
Rule 4 :	(M, M)	:	Two missionaries sailing the boat from bank-2 to bank-1
Rule 5 :	(M, C)	:	One missionary and one Cannibal sailing the boat from bank-1 to bank-2
Rule 6 :	(C, M)	:	One missionary and one Cannibal sailing the boat from bank-2 to bank-1
Rule 7 :	(C, C)	:	Two Cannibals sailing the boat from bank-1 to bank-2
Rule 8 :	(C, C)	:	Two Cannibals sailing the boat from bank-2 to bank-1
Rule 9 :	(0, C)	:	One Cannibal sailing the boat from bank-1 to bank-2
Rule 10 :	(C, 0)	:	One Cannibal sailing the boat from bank-2 to bank-1

All or some of these production rules will have to be used in a particular sequence to find the solution of the problem. The rules applied and their sequence is presented in the following Table 2.2.

Table 2.2: Rules applied and their sequence in Missionaries and Cannibals problem

After application of rule	persons in the river bank-1	persons in the river bank-2	boat position
Start state	M, M, M, C, C, C	0	bank-1
5	M, M, C, C	M, C	bank-2
2	M, M, C, C, M	C	bank-1
7	M, M, M	C, C, C	bank-2
10	M, M, M, C	C, C	bank-1
3	M, C	C, C, M, M	bank-2
6	M, C, C, M	C, M	bank-1
3	C, C	C, M, M, M	bank-2
10	C, C, C	M, M, M	bank-1
7	C	M, M, M, C, C	bank-2
10	C, C	M, M, M, C	bank-1
7	0	M, M, M, C, C, C	bank-2

2.5.7 Tower of Hanoi Problem

This is a historic problem. It can be described as follows:

> *"Near the city of 'Hanoi', there is a monastery. There are three tall posts in the courtyard of the monastery. One of these posts is having sixty-four disks, all having a hole in the centre and are of different diameters, placed one over the other in such a way that always a smaller disk is placed over the bigger disk. The monks of the monastery are busy in the task of shifting the disks from one post to some other, in such a way that at no point of time, a bigger disk is placed above smaller disk. Only one disk can be removed at a time. Moreover, at every point of time during the process, all other disks than the one removed, should be on one of the posts. The third post can be used as a temporary resting place for the disks. We have to help the monks in finding the easiest and quickest way to do so".*

Can we really help the monks? Before indulging ourselves in finding the solution of this problem, let us realize that even the quickest method to solve this problem might take longer time than the time left for the whole world to finish! Are you still interested in attempting the problem?

64-disk configuration is shown in Fig 2.8.

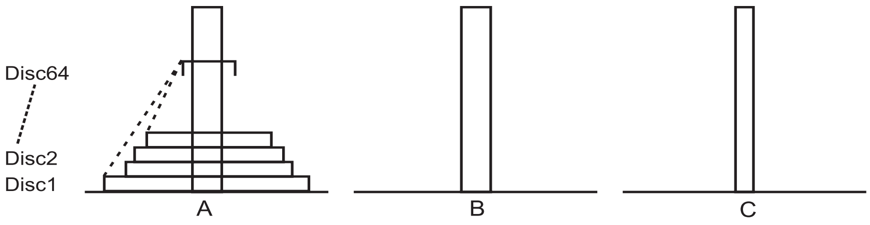


Fig 2.8: Tower of Hanoi problem

2.5.8 Traveling Salesperson Problem

This problem falls in the category of path finding problems. The problem is defined as follows:

> *"Given 'n' cities connected by roads, and distances between each pair of cities. A sales person is required to travel each of the cities exactly once. We are required to find the route of salesperson so that by covering minimum distance, he can travel all the cities and come back to the city from where the journey was started".*

Diagrammatically, it is shown in Fig 2.9.

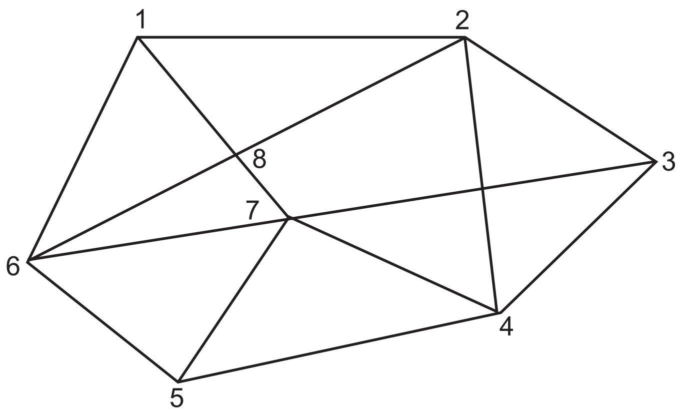


Fig 2.9: Cities and paths connecting these

The basic travelling salesperson problem comprises of computing the shortest route through a given set of cities.

Following Table 2.3 shows number of cities and the possible routes mentioned against them.

Table 2.3: Possible routes of travelling salesperson problem

Number of cities	Possible Routes
1	1
2	1 -2-1
3	1 -2 -3 1
	1 -3 -2 1
4	1- 2- 3- 4-1
	1- 2- 4- 3- 1
	1- 3- 2- 4- 1
	1- 3- 4- 2- 1
	1- 4- 2- 3-1
	1- 4- 3- 2- 1

We can notice from here that the number of routes between cities is proportional to the factorial of the (number of cities – 1), i.e., for three cities, the number of routes will be equal to !2 (2x1), and for four cities, !3 (3x2x1).

While there are !9 = 362880 routes for 10 cities, there are !29 = 8.8 E30 possible routes for 30 cities. The travelling salesperson problem is a classic example of ***combinatorial explosion***, because the number of routes increases so rapidly that there are no practical solutions for realistic number of cities. If it takes 1 hour of a mainframe CPU time to solve for 30 cities, it will take 30 hours for 31 cities and 330 hours for 32 cities. Practically, the actual application of such type of problem is for routing a data packet on computer networks. It requires managing thousands of cities. Hence for this much large amount of data the required problem solution time will be unaffordable.

The solution of this problem is done using neural network. The neural network can solve the 10 city case just as fast as the 30 city case whereas a conventional computer takes much longer time for these.

2.5.9 Magic Square

The problem is represented as follows:

> *"We are given a square of same number of rows and columns, where consecutive numbers are filled in blank spaces of each row and column such that numbers in each row, column and two diagonals, add up to the same number. The numbers have to be consecutive and one number can be used only once".*

Therefore, the 3×3 square and 4×4 square would require any set of 9 and 16 consecutive numbers respectively. The problem is finding out the placement of numbers. The magic squares of order 3×3, where numbers from 1 to 9 are filled, is shown in Fig 2.10. It should be noted that there could be multiple solutions of magic square problem.

6	7	2
1	5	9
8	3	4

Fig. 2.10: A 3×3 magic square configuration

Let us describe the manual process of solving 3×3 magic square by filling numbers from 1 to 9. The steps required to attempt the problem are presented as follows:

Step 1: The sum of the numbers in each row and column can be predetermined as:

$$\text{Sum of numbers in each row, column and diagonal} = \frac{\text{Sum of numbers used}}{\text{Number of rows or columns}}$$

In our case, this figure is equal to 45/3, i.e. 15. (Sum of numbers from 1 to 9 divided by 3)

Step 2: Suppose the numbers to be filled in each blank space are x_1 to x_9. The magic square then will look like:

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

Now, since we know the sum of numbers in each row, column and diagonals is 15, we can form following equations:

$x_1 + x_2 + x_3 = 15$ .... (1)
$x_4 + x_5 + x_6 = 15$ .... (2)
$x_7 + x_8 + x_9 = 15$ .... (3)
$x_1 + x_4 + x_7 = 15$ .... (4)
$x_2 + x_5 + x_8 = 15$ .... (5)
$x_3 + x_6 + x_9 = 15$ .... (6)
$x_1 + x_5 + x_9 = 15$ .... (7)
$x_3 + x_5 + x_7 = 15$ .... (8)

Step 3: We will have to solve the above equations for finding the values of numbers. You may notice that number x_5 comes in most of the equations hence; we should try to find the value of x_5 first.

Add the equations (5), (7) and (8), to get:

$x_2 + x_5 + x_8 + x_1 + x_5 + x_9 + x_3 + x_5 + x_7 = 45$

Putting values from equations (1) and (3), we get,

$3x_5 = 15 \Rightarrow x_5 = 5$.

Step 4: Put the value of x_5 in the equations where it appears, we get,

$x_4 + x_6 = 10$ .... (9)
$x_2 + x_8 = 10$ .... (10)
$x_1 + x_9 = 10$ .... (11)
$x_3 + x_7 = 10$ .... (12)

Step 5: Assume $x_1 = 1$, hence $x_9 = 9$ [from equation (11)]

After getting three values, our magic square will look like:

1	x_2	x_3
x_4	5	x_6
x_7	x_8	9

Now, try to find the location of number 2. Any number out of x_2, x_3, x_4, and x_7 cannot be 2, because in that case remaining number will be 12 to make the sum as 15, which is not permissible. Hence, either x_6 or x_8 will be 2. But, it is also not possible, because in that case, x_3 or x_7 will be 4 and x_2 or x_4 will become 10, which is again not possible. Therefore, x_1 cannot be 1. Revise this step with another assumption.

Step 6: Assume $x_2 = 1$, hence $x_8 = 9$ [from equation (10)].

After getting three values, our magic square will look like drawn ahead.

x_1	1	x_3
x_4	5	x_6
x_7	9	x_9

Now, any number out of x_1 and x_3 cannot be 2, because of the reason mentioned in the above step, hence one number out of x_4, x_6, x_7 and x_9 will be equal to 2. Assume $x_4 = 2$, hence $x_6 = 8$ [from equation (9)]. Notice that 3 can not come in the same row, column or diagonal in which either 1 or 2 comes, because then, the remaining number will be either 11 or 10, which is not possible. Hence, for $x_4 = 2$, x_1, x_3, and x_7 cannot be 3, resulting in $x_9 = 3$. However, this is also not possible, because number 3 cannot come with 9 since the remaining number in that case will also be 3, which cannot be. Therefore, x_2 or x_4 cannot be 2. Try another assumption.

Step 7: Assume x_7 or $x_9 = 2$, resulting in x_3 or $x_1 = 8$ [from equations (12) and (11)].

After getting one more number, our magic square will look like:

x_1	1	x_3
x_4	5	x_6
2	9	x_9

x_1	1	x_3
x_4	5	x_6
x_7	9	2

From the above squares, we can calculate the remaining numbers as, $x_9 = 4$, $x_1 = 6$, $x_3 = 8$, $x_6 = 3$ and $x_4 = 7$ for the first square; and $x_7 = 4$, $x_3 = 6$, $x_1 = 8$, $x_4 = 3$ and $x_6 = 7$ for the second square; and the solutions for the magic square will be:

6	1	8
7	5	3
2	9	4

8	1	6
3	5	7
4	9	2

As mentioned earlier, there can be several solutions to the magic square depending upon which number you assume as 1 to start with in the step 5.

Similarly, we can solve magic square of the order 4 × 4. A typical solution is given as follows in the Fig 2.11 :

7	13	12	2
10	4	5	15
1	11	13	8
16	6	3	9

Figure 2.11: Magic square of the order 4 x 4

2.5.10 Language Understanding Problems

This type of problems include the understanding of natural languages, conversion of one language to another language, language comprehension, design of intelligent natural language interfaces etc. It also includes answering of a query using a database. More regarding such type of problems and other various issues involved in natural language understanding are deal with in another chapter of this book.

2.5.11 Monkey and Banana Problem

This problem is described as follows:

> *"A monkey and a bunch of banana are present in a room. The bananas are hanging from the ceiling. The monkey cannot reach the bananas directly. However, in the room there is one chair and a stick. The monkey can reach the banana standing on the chair. We have to find the sequence of events by which monkey can reach the bananas".*

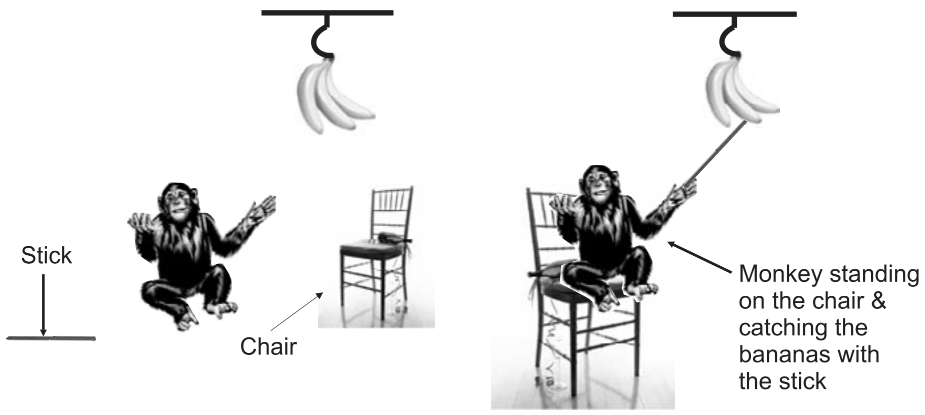

Stick
Chair
Monkey standing on the chair & catching the bananas with the stick

Fig 2.12: Monkey and banana problem

Solution of this problem means finding the sequence of actions for the monkey to reach the banana. It is much simpler than the problems of water and jug or Missionaries and Cannibals discussed above, hence, we leave it for the readers to formulate the set of production rules and to find the appropriate sequence of the actions required for the solution. The problem is pictorially represented in Fig 2.12.

2.5.12 Cryptarithmatic Puzzle

It is a puzzle involving decoding of digit represented by a character. It is in the form of some arithmetic equation where digits are distinctly represented by some characters. The problem requires finding of the digit represented by each character. One such problem is shown in Fig. 2.13.

$$\begin{array}{r} \text{R I N G} \\ +\ \text{D O O R} \\ \hline \text{B E L L} \end{array}$$

Fig. 2.13: A cryptarithmatic problem

These types of problems require constraint satisfaction. Constraints are that all the laws of arithmetic must hold good and any letter should represent same digit wherever it comes in the puzzle. Also, no two different letters can be represented by same digit. For example, in the puzzle shown above, the laws of summation of two given numbers must hold good and digit R, which comes at two different places must be represented by same digit. Similarly, letter O and L must be represented by same digit, however, R, O and L, together with other letters, must be represented by different digits. The process involved in solving these puzzles is described in detail in the next chapter of this book.

2.5.13 Block World Problems

The problem can be represented as follows:

> *"There are some cubic blocks, out of which, each is placed either on the table or on other block forming a particular configuration. We have to move the blocks to arrange them to form some other given configuration by applying minimum number of moves. The requirement of moving a block is that only the top block from a group can be shifted in a move. It can be placed either on the table or on top of any other block."*

The live application of this type of problem can be seen in a container depot where lot of containers are placed one over the other in several groups. A crane or a robot is used to arrange these containers in different groups for loading in trains and ships. The start and goal states of a typical block world problem are shown in Fig 2.14.

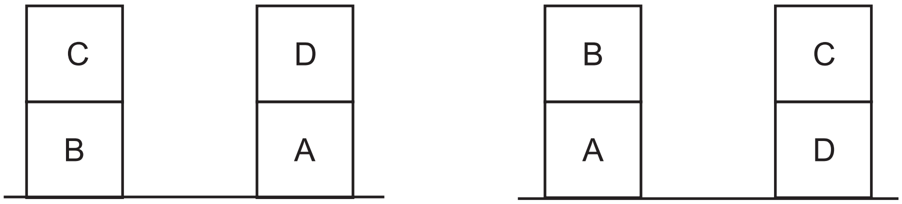


Fig. 2.14: The start and goal configuration of block world problem

Detailed account of this problem is given in the section of *Planning* of chapter -11 of this book.

2.6 NATURE OF AI PROBLEMS

In this section, we will highlight the nature of real world AI problems. As mentioned earlier, problems can be of several types and solution of a particular problem depends upon the nature of the problem. We may categorize problems based on their nature in the manner discussed hereunder:

2.6.1 Path Finding Problems

In this type of problems, the solution involves reporting of path (or sequence of steps used to obtain the solution). The traveling salesperson problem discussed above is an example of this kind of problem.

2.6.2 Decomposable Problems

In this type of problems, the problem can be broken into small subproblems and the solution of the main problem is obtained by joining the solutions of small subproblems, e.g. in mathematical equation, if the expression involves the solution of various components then these components can be solved independently, and the complete answer can be obtained by joining the solutions of subproblems. However, all the real world problems are not decomposable problems.

Consider following arithmetic equation:

$$\int (x^3 + 3x^2 + \sin^2 x + \cos^2 x)\, dx = 0$$

If we want to solve this equation, we can find the solution of different integrals separately and then join the results to get complete solution. Hence, above problem is broken into four problems, i.e. (i) $\int x^3\, dx$, (ii) $\int 3x^2\, dx$, (iii) $\int \sin^2 x\, dx$, (iv) $\int \cos^2 x\, dx$.

2.6.3 Recoverable Problems

There are certain types of problems, where the application of operator can be reverted if required, and on the initial state, a new operator can be applied. Such types of problems are called recoverable problems. 8-puzzle problem is an example of this kind because if you apply a move and find that it is wrong or not worth, you may revert to the original position and apply another move. The recoverability of a problem plays an important role in determining the control structure necessary for problem solution. There is another type of problems like "theorem proving" where the steps can be ignored. These are called ignorable problems. These can be solved using simple control structures, which are easy to implement. But there are certain types of problems like medical diagnosis problems where if an operator is applied, it cannot be reverted to original state. In such types of problems, the selection of operator is very critical because if a wrong operator is applied, it may be possible that a solution is never obtained even if it exists. Such types of problems are called '**irrecoverable**' problems. Chess playing is another irrecoverable problem where once you apply a move, you are not allowed to revert. The recoverable problems require little complex control structure, which checks the outcomes of every move and in case the move is found to be wrong, it is reverted and initial state is restored back. The control structure of irrecoverable problems is difficult because it will involve a system that expands a great deal of effort in exploring each decision, since the decisions once chosen must be final.

2.6.4 Predictable Problems

These are the types of problems where all the outcomes of a particular move can be judged with definiteness. For example, consider the 8-puzzle problem where the outcomes after applying a certain move can be easily predicted because all the numbers and their positions are open before you. Hence, while deciding about a move, the suitability of these outcomes can be judged and planning the entire sequence of moves is possible. Whereas in card games like bridge or game where all the cards are not open, it cannot be decided in advance that what will be the effect of a particular move because your next move depends entirely upon the move of other players involved, hence planning a sequence of moves is not possible. Similar situation will be there in a game involving two players. One player can not judge the move played by other player. In such situations, multiple moves are considered for suitability and a probabilistic approach is applied to decide about the best move.

2.6.5 Problems Affecting the Quality of Solution

There are certain types of AI problems where the process of finding solution stops by just finding one solution and there is no need to ensure the validity of this solution by finding the other solutions, e.g., the database query applications. In query applications, whenever the query is answered, the other possible answers are not checked. However, in route finding problems (like traveling salesperson problem) once a route from source to destination is obtained, still other possibilities need to be checked to judge whether it is the shortest path or not (optimal solution). Thus, we can say that there are certain types of problems where any one solution is enough and in other types of problems where to accept a solution all the possible solutions need to be checked.

2.6.6 State Finding Problem

In this type of AI problems, the answer is reporting of a state instead of path. The examples of such type of problems are natural language understanding applications. In such type of applications the interpretation of a particular sentence is required and it is not important how the solution is obtained. As far as the interpretation of the input sentence is correct, the steps used in obtaining the solution can be ignored. However, there are another kinds of problems where the solution demands the reporting of state as well as path. These are the problems requiring justification, e.g., medical diagnosis problems using expert system. Here, besides the name of the medicine, the patient requires explanation about the process or the path applied for finding that particular medicine. Thus, it involves the 'belief' of user as well. To satisfy him, the process of obtaining the solution (i.e. the path) and the solution (i.e. the state) both need to be stored. Hence, the types of problems, which involve human belief, fall in the category of problems where state and path both should be reported.

2.6.7 Problems Requiring Interaction

This is the kind of problems, which require asking some questions from the user or interacting with the user. There are many AI programs already built which interact with the user very frequently to provide additional input to the program and to provide additional reassurance to the user. In the situations where computer is generating some advice for a particular problem, it is always advisable to take the views of user at regular interval. Doing this will help providing the output in the form suitable to the user.

2.6.8 Knowledge Intensive Problems

These are kinds of AI problems, which require large amount of knowledge for their solution. Consider the tic-tac-toe game. Here, to play the game, the required amount of knowledge is only about legal moves and other player's moves. This is little amount of knowledge. Whereas in chess game, the number of legal moves are more, and to decide a particular move, more future moves of opponent and those of oneself in "*look ahead manner*" (technically called *ply*) are visualized. Hence, the amount of knowledge required to play a chess game is more as compared to tic-tac-toe game. Further, if we consider the medical expert system, the amount of knowledge required is enormous. Such types of AI applications are called knowledge intensive applications. Similarly, in the application of a data query, the knowledge required is only about the database and query language whereas if it is a natural language understanding program, then it will require vast amount of knowledge comprising of syntactic, semantic and pragmatic knowledge. Hence, the amount of knowledge and role of knowledge varies in different AI problems.

From the above discussion, it is evident that the nature of various AI problems is widely different and in order to choose the most appropriate method for solving a particular problem, the nature of problem needs to be analyzed. The problem characteristics can be summarized as follows:

1. If the problem is decomposable into independent smaller or easier sub problems.
2. Is backtracking possible or not.
3. Is the problem's universe predictable.
4. Is a good solution to the problem obvious without comparison to all other possible solutions.
5. Is the desired solution a state or a path.
6. Is a large amount of knowledge absolutely required to solve the problem or is knowledge important only to constrain the search.

2.7 SEARCH TECHNIQUES

As stated earlier, the process of the solution of AI problem searches the path from start state to goal state. This is very important aspect of problem solving because search techniques not only help in finding most viable path to reach the goal state, but also make the entire process efficient and economical. In this section, we discuss basic search techniques adopted for finding the solution of AI problems. Broadly, the search is of two types:

(i) *Blind (or unguided or uninformed) search*: The uninformed or blind search is the search methodology having no additional information about states beyond that provided in the problem definitions. In this search total search space is looked for solution.

(ii) *Heuristic (or guided or informed) search*: These are the search techniques where additional information about the problem is provided in order to guide the search in a specific direction.

In this chapter, we will discuss the unguided search techniques and the guided search techniques will be dealt with in the next chapter. Following search techniques comes under the category of unguided search techniques:

(i) Breadth-First search (BFS)
(ii) Depth-First search (DFS)
(iii) Depth-Limited search (DLS)
(iv) Bidirectional search

2.7.1 Breadth-First Search

In this type of search, the state space is represented in form of a tree. In addition, the solution is obtained by traversing through the tree. The nodes of tree represent start state, various intermediate states and the goal state. While searching for the solution, the root node is expanded first, then all the successors of the root node are expanded, and in next step all successors of every node are expanded. The process continues till goal state is achieved, e.g., in the tree shown in following Fig. 2.15, the nodes will be explored in order A, B, C, D, E, F, G, H, I, J, K, L:

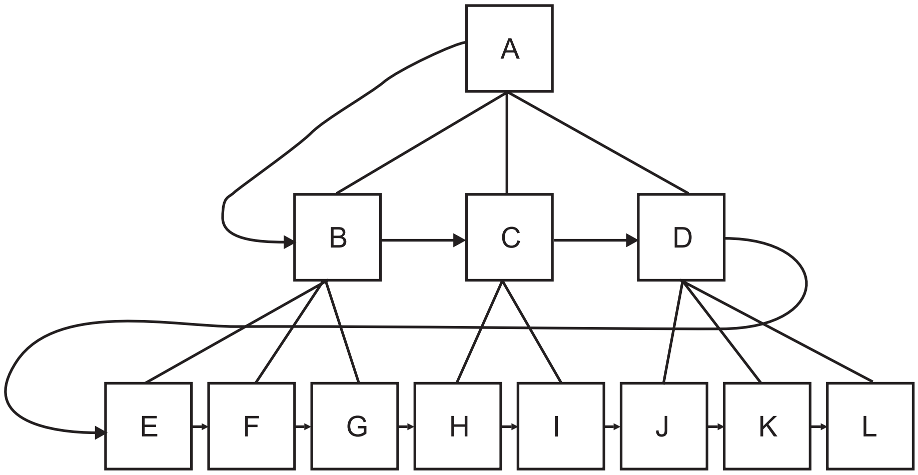


Fig. 2.15: Breadth-First Search Tree

In breadth-first search, the space complexity is more critical as compared to time complexity. Analysis shows that the time and memory requirement of the problem of depth 8 and 10 are 31 hours, 1 terabyte and 129 days, 101 terabyte respectively. Fortunately, there are other strategies taking lesser time in performing the search. In general, the problems involving search of exponential complexity (like chess game) cannot be solved by uninformed methods for the simple reason, the size of the data being too big. The data structure used for breadth first search is First In First Out (FIFO).

Algorithm for breadth first search is described as follows:

```
(i) Create a variable called Node-List and set it to initial state.
(ii) Until a goal state is found or Node-List is empty do:
    (a) Remove the first element from node-list and call it E. If node-list was
        empty, quit.
    (b) For each way that each rule can match the state described in E do:
        (i) Apply the rule to generate a new state.
        (ii) If new state is a goal state, quit and return this state.
        (iii) Otherwise, add new state to the end of node-list.
```

2.7.1.1 Advantages of Breadth First Search (BFS)

The breadth first search is not caught in a blind alley. This means that, it will not follow a single unfruitful path for very long time or forever, before the path actually terminates in a state that has no successors. In the situations where solution exists, the breadth first search is guaranteed to find it. Besides this, in the situations where there are multiple solutions, the BFS finds the minimal solution. The minimal solution is one that requires the minimum number of steps. This is because of the fact that in breadth first search, the longer paths are never explored until all shorter ones have already been examined. Thus, if the goal state is found during the search of shorter paths, longer paths would not be required to be searched, saving time and efforts.

Traveling sales person problem discussed above can be solved using Breadth-First Search technique. It will simply explore all the paths possible in the tree and will ultimately come out with the shortest path desired. However, this strategy works well only if the number of cities is less. If we have large number of cities in the list, it fails miserably because number of paths and hence the time taken to perform the search become too big to be controlled by this method efficiently.

2.7.2 Depth First Search

There can be another type of search strategy than the one described above. In this type of approach, instead of probing the width, we can explore one branch of a tree until the solution is found or we decide to terminate the search because either a dead end is met, some previous state is encountered or the process becomes longer than the set time limit. If any of the above situations is encountered and the process is terminated, a backtracking occurs. A fresh search will be done on some other branch of the tree, and the process will be repeated until goal state is reached. This type of technique is known as Depth-First Search technique.

It generates the left most successor of root node and expands it, until dead end is reached. The search proceeds immediately to the deepest level of search tree, or until the goal is encountered. The sequence of explored nodes in the previous tree will be A, B, E, F, C, G, H, K, L, D, I, J. The search is shown in Fig 2.16 below:

The DFS has lesser space complexity, because at a time it needs to store only single path from root to leaf node. The data structure used for DFS is Last-In-

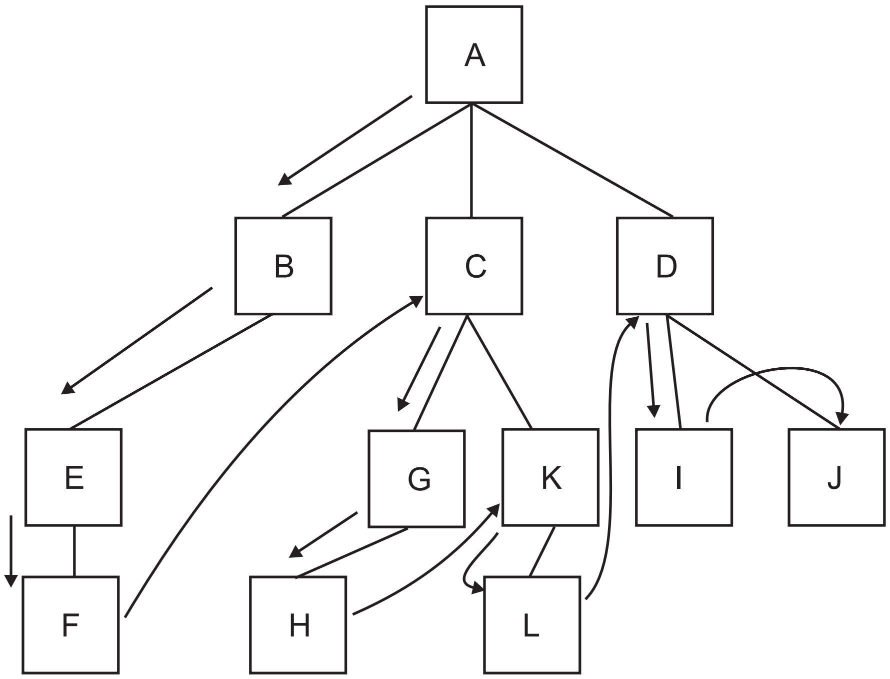


Fig 2.16: Depth First search tree

First-Out (LIFO).

Algorithm for Depth First Search is described as follows:

1. If initial state is a goal state, quit and return success.
2. Otherwise, do the following until success or failure is reported:
   (a) Generate a successor E of the initial state. If there are no more successors signal failure.
   (b) Call Depth-First Search with E as initial state.
   (c) If success is obtained, return success, otherwise continue in this loop.

The procedure to perform DFS for tree of node ‘n’ is given below:

```
Procedure: unbounded  DFS for tree of node n, [ DFS1(n)].
Start
If n is a goal node then
Start

Solution : = n;
Exit;
End;
For each successor ni of n do
If ni is not an ancestor  of n then
                          P1(ni);
```

```
DDFS (root, d);
Until success;
End;
```

2.7.4 Bidirectional Search

As the name indicates, it is search in two directions. In the search methods discussed above, the search proceeds only from start to goal, unlike in bidirectional search, where two simultaneous searches are done. One search proceeds in forward direction (i.e. starting from start node towards goal node) and another search proceeds in the backward direction (i.e. starting from goal node towards start node). Wherever two searches meet, the process stops. The bidirectional search is advantageous then unidirectional search, because for a tree of depth ‘d’ and branching factor ‘b’, the unidirectional search requires b^d number of comparisons but bidirectional search requires $b^{d/2}$ comparisons in each direction. As $b^{d/2} + b^{d/2} < b^d$, the bidirectional search has lesser time complexity. However, it is applicable to only on those situations where search tree is completely defined, i.e. all actions in state space are reversible. A bidirectional search is shown in Fig 2.17.

Here A, B, C, E, F are cities to be traveled and the distance between each pair of cities is mentioned in the graph. In the solution of this problem, the location of cities can be represented in graph by nodes indicating the cities. The starting state

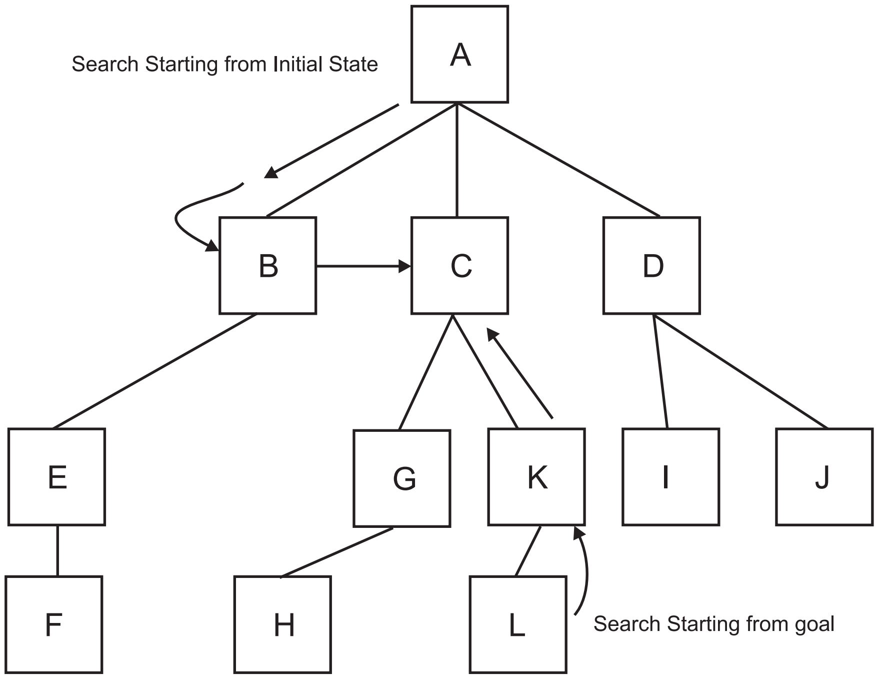


Fig. 2.17: Example of Bidirectional search