

Artificial Intelligence

Module1

Contents

Chapter#	Title	Page No
1	<i>What is Artificial Intelligence?</i>	2
2	<i>Problem Spaces and search,</i>	14
3	<i>Heuristic search techniques</i>	29

Dr.TGS

Chapter 1: What is Artificial Intelligence?

Q1. What is Intelligence?

Answer:

Intelligence is a mental capability that involves the ability to understand reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience. It is not merely book learning, a narrow academic skill, or test-taking smarts. Rather, it reflects a broader and deeper capability for comprehending our surroundings—"catching on," "making sense" of things, or "figuring out" what to do.

According to **Niccolo Machiavelli**, there are three kinds of intelligence: one kind understands things for itself, the other appreciates what others can understand, the third understands neither for itself nor through others. This first kind is excellent, the second kind good and the third kind useless.

Q2. What is Artificial Intelligence? How it is Accomplished?

Answer:

According to **Elaine Rich and Kevin Knight**,

"Artificial Intelligence (AI) is the study of how to make computers do things, which at the moment people do better".

Artificial intelligence (AI) is an area of computer science that emphasizes the creation of intelligent machines that work and reacts like humans. Artificial Intelligence involves mainly two things: **1.** Studying the thought process of humans and **2.** Dealing with representing those processes via machines. Some of the activities computers with artificial intelligence are designed for include: ***Speech recognition, Learning, Planning and Problem solving.***

AI is accomplished by studying how human brain thinks and how human learn, decide and work while trying to solve a problem and then using the outcomes of this study as a basis of developing intelligent software and systems.

Q3. What are the advantages of Artificial Intelligence?

Answer: It is clear that AI will have to fill the gap of human knowledge and it will make man's work easier and pleasant. However, some presumed advantages of using AI are given below.

1. **AI can replace humans** being in some more specific jobs for some business store and household day to day activities and will help to sort out the manpower.
2. **AI machines can help in hospitals**, providing food and medicines where human being feared to be attacked by such disease. Robotics is good example.
3. **It is estimated that AI will help human being in aeronautics** to know the universe.
4. **AI can help to unfold the fields which are not explored yet.**
5. **By using AI we can send machines in such areas where human being loss is estimated.** E.g. Military and Terrorism.

Q4. Discuss about the different areas that AI problems and work are focused? Give the categories of *tasks that are targets of work in AI.*

Answer: Much of the early AI problems and work focused on formal tasks such as ***Game Playing and Theorem Proving***. Another early foray into AI focused on the sort of the problem solving that we do every day when we decide how to get to work in the morning, often called ***Common Sense Reasoning***. It includes reasoning about physical objects and their relationships to each other as well as reasoning about actions and their consequences. As ***AI research*** progressed and techniques for handling larger amounts of world knowledge were developed some progress was made on the tasks like perception (***vision and speech***), *natural language understanding and problem solving in specialized domains such as medical diagnosis and chemical analysis*. In addition to these ***mundane tasks AI programs can also be used to solve the problems which require expertise*** like Engineering design, scientific discovery, medical diagnosis and financial planning. Figure below categories of the tasks that are the targets of work in AI.

Mundane Tasks	Formal Tasks	Expert Tasks
<ul style="list-style-type: none"> • Perception <ul style="list-style-type: none"> ○ Vision ○ Speech • Natural Language <ul style="list-style-type: none"> ○ Understanding ○ Generation ○ Translation • Common Sense Reasoning • Robot Control 	<ul style="list-style-type: none"> • Games <ul style="list-style-type: none"> ○ Chess ○ Backgammon ○ Checkers- Go • Mathematics <ul style="list-style-type: none"> ○ Geometry ○ Logic ○ Integral Calculus ○ Proving Properties of Programs 	<ul style="list-style-type: none"> • Engineering <ul style="list-style-type: none"> ○ Design ○ Fault Finding ○ Manufacturing Planning • Scientific Analysis • Medical Diagnosis • Financial Analysis

Q5. What are Our Underlying Assumptions about Intelligence?

Q5. Explain Physical Symbol Hypothesis.

Answer: At the heart of research in Artificial Intelligence lies what Newell and Simon [1976] call the **physical symbol hypothesis**. They define a physical symbol system as follows:

The physical symbol system consists of a set of entities called symbols which are physical patterns that can occur as components of another entity called symbol structure (expression). Besides these structures the system also contains a collection of processes that operate on expressions to produce other expressions: processes of creation, modification, reproduction and destruction. A physical symbol system is a machine that produces through time and evolving collection of symbol structures. Such a system exists in a world of objects wider than just this symbolic expression themselves.

The Physical Symbol System Hypothesis: A physical symbol system has the necessary and sufficient means for general intelligent action.

Q6. List out the less desirable properties of the knowledge.

Answer: Intelligence requires knowledge. In the real world, the knowledge possesses some less desirable properties including:

- a. It is voluminous next to unimaginable
- b. It is hard to characterize accurately
- c. It is not well organized or well formatted
- d. It is constantly changing

Q7. What kinds of techniques will be useful for solving AI PROBLEMS?

Answer: **AI technique** is manner /method that exploits organizes and uses the knowledge efficiently and represent in such a way that:

- The knowledge captures generalization
- It can understand by the people who provide it.
- It can be easily modified to correct errors and to reflect changes in the world and in our world view.
- It can be used in a great many situation if it is not totally accurate or complete
- It can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities.

Q8. Discuss the different approaches to solve the problem of Tic-Tac-Toe.

Answer:

The **three programs** or approaches to play tic-tac-toe are discussed below. The programs in this series increase in:

- Their complexity
- Their use of generalizations
- The clarity of their knowledge
- The extensibility of their approach

Thus, they move toward being representations of what we call AI techniques.

Program 1

Data Structures	
Board	A nine-element vector representing the board, where the elements of the vector correspond to the board positions as follows: An element contains the value <ul style="list-style-type: none"> • if the corresponding square is blank, • if it is filled with an X, or • if it is filled with an O.
Movetable	A large vector of 19,683 elements (3^9), each element of which is a nine-element vector. The contents of this vector are chosen specifically to allow the algorithm to work.
Algorithm	
<p>To make a move, do the following:</p> <ol style="list-style-type: none"> 1. View the vector Board as a ternary (base three) number. Convert it to a decimal number. 2. Use the number computed in step 1 as an index into Movetable and access the vector stored there. 3. The vector selected in step 2 represents the way the board will look after the move that should be made. So set Board equal to that vector. 	
Comments : This program is very efficient in terms of time . But it takes a lot of space to store the table that specifies the correct move to make from each board position. Possibility of errors.	

Program 2

Data Structures	
Board	A nine-element vector representing the board, as described for Program 1 . But instead of using the numbers 0, 1 , or 2 in each element, we store 2 (indicating blank), 3 (indicating X), or 5 (indicating O).
Turn	An integer indicating which move of the game is about to be played; 1 indicates the first move, 9 the last.
Algorithm : The main algorithm uses three subprocedures:	
Make2	Returns 5 if the center square of the board is blank, that is, if Board[5]= 2. Otherwise, this function returns any blank no corner square(2, 4, 6, or 8).
Posswin(<i>p</i>)	Returns 0 if player <i>p</i> cannot win on his next move; otherwise, it returns the number of the square that constitutes a winning move. This function will enable the program both to win and to block the opponent's win. Posswin operates by checking, one at a time, each of the rows, columns, and diagonals. Because of the way values are numbered, it can test an entire row (column or diagonal) to see if it is a possible win by multiplying the values of its squares together. If the product is 18 ($3 \times 3 \times 2$), then X can win. If the product is 50 ($5 \times 5 \times 2$), then O can win. If we find a winning row, we determine which element is blank, and return the number of that square.
Go(<i>n</i>)	Makes a move in square <i>n</i> . This procedure sets Board[<i>n</i>] to 3 if Turn is odd, or 5 if Turn is even. It also increments Turn by one. The algorithm has a built-in strategy for each move it may have to make. It makes the odd-numbered moves if it is playing X, the even-numbered moves if it is playing O.
The strategy for each turn is as follows:	

Turn=1 Go(1) (upper left comer).
 Turn=2 If Board[5] is blank, Go(5), else Go(1).
 Turn=3 If Board[9] is blank, Go(9), else Go(3).
 Turn=4 If Posswin(X) is not 0, then Go(Posswin(X))» [i.e., block opponent's win], else Go(Make2).
 Turn=5 If Posswin(X) is not 0 then Go(Posswin(X)) [i.e., win] else if Posswin(O) is not 0, then Go(Posswin(O)) [i.e., block win], else ifBoard[7] is blank, then Go(7), else Go(3). [Here the program is trying to make a fork.]
 Turn=6 If Posswin(O) is not 0 then Go(Posswin(O)), else if Posswin(X) isnot 0, then Go(Posswin(X)), else Go(Make2).
 Turn=7 If Posswin(X) is not 0 then Go(Posswin(X), else if Posswin(O) isnot 0, then Go(Posswin(O)), else go anywhere that is blank.
 Turn=8 If Posswin(O) is not 0 then Go(Posswin(O)), else if Posswin(X) is not0,then Go(Posswin(X)), else go anywhere that is blank.
 Turn=9 Same as Turn=7

Comments : This program is not quite as efficient in terms of time as the first one since it has to check several conditions before making each move. But it is a lot more efficient interms of space. It is also a lot easier to understand the program's strategy or to change the strategy if desired.

Program 2'

This program is identical to Program 2 except for one change in the representationof the board. We again represent the board as a nine-element vector, but this time weassign board positions to vector elements as follows:

8	3	4
1	5	9
6	7	2

Notice that this numbering of the board produces a magic square: ***all the rows, columns, and diagonals sum to 15***. This means that we can simplify the process of checking for a possible win. To check for a possible win for one player, we consider each pair of squares owned by that player and compute the difference between 15 and the sum of the two squares. If this difference is not positive or if it is greater than 9, then the original two squares were not collinear and so can be ignored. Otherwise, if the square representing the difference is blank, a move there will produce a win. Since no player can have more than four squares at a time, there will be many fewer squares examined using this scheme than there were using the more straightforward approach of Program 2. This shows how the choice of representation can have a major impact on the efficiency of a problem-solving program.

Comments : More easier and efficient approach.

This comparison raises an interesting question about the relationship between the way people solve problems and the way computers do. Why do people find the row-scan approach easier while the number-counting approach is more efficient for a computer? We do not know enough about how people work to answer that question completely. One part of the answer is that people are parallel processors and can look at several parts of the board at once, whereas the conventional computer must look at the squares one at a time. Sometimes an investigation of how people solve problems sheds great light on how computers should do so. At other times, the differences in the hardware of the two seem so great that different strategies seem best.

Program 3

Data Structures	
Board Position	A structure containing a nine-element vector representing the board, a list of board positions that could result from the next move, and a number representing an estimate of how likely the board position is to lead to an ultimate win for the player to move.
Algorithm :To decide on the next move, look ahead at the board positions that result from each possible move. Decide which position is best (as described below), make the move that leads to that position, and assign the rating of that best move to the current position. To decide which of a set of board positions is best, do the following for each of them: <ol style="list-style-type: none">1. See if it is a win. If so, call it the best by giving it the highest possible rating.2. Otherwise, consider all the moves the opponent could make next. See which of them is worst for us (by recursively calling this procedure). Assume the opponent will make that move. Whatever rating that move has, assign it to the node we are considering.3. The best node is then the one with the highest rating. This algorithm will look ahead at various sequences of moves in order to find a sequence that leads to a win. It attempts to maximize the likelihood of winning, while assuming that the opponent will try to minimize that likelihood. This algorithm is called the <i>minimax procedure</i> , and it is discussed in detail in later.	
Comments: <i>This program will require much more time than either of the others since it must search a tree representing all possible move sequences before making each move. But it is superior to the other programs in one very big way: It could be extended to handle games more complicated than tic-tac-toe, for which the exhaustive enumeration approach of the other programs would completely fall apart.</i> It can also be augmented by a variety of specific kinds of knowledge about games and how to play them..	

Q8. Discuss the different approaches to solve the problem of Question Answering.

Answer: The three different approaches or programs used to read in English text and then answer questions, is discussed below. Consider the following text:

Mary went shopping for a new coat. She found a red one she really liked. When she got it home, she discovered that it went perfectly with her favorite dress.

The three approaches discussed here will attempt to answer each of the following questions:

Q1: What did Mary go shopping for?

Q2: What did Mary find that she liked?

Q3: Did Mary buy anything?

Program 1: This program attempts to answer questions using the literal input text. It simply matches text fragments in the questions against the input text.

Data Structures	
Question Patterns	A set of templates that match common question forms and produce patterns to be used to match against inputs.
Text	The input text stored simply as a long character string.
Question	The current question also stored as a character string.
The Algorithm	
<p>To answer a question, do the following:</p> <ol style="list-style-type: none"> 1. Compare each element of Question Patterns against the Question and use all those that match successfully to generate a set of text patterns. 2. Pass each of these patterns through a substitution process that generates alternative forms of verbs so that, for example, "go" in a question might match "went" in the text. This step generates a new, expanded set of text patterns. 3. Apply each of these text patterns to Text, and collect all the resulting answers. 4. Reply with the set of answers just collected. <p>Example : The template "What did x v" matches this question and generates the text pattern "Mary go shopping for z." After the pattern-substitution step, this pattern is expanded to a set of patterns including "Mary goes shopping for z," and "Marywent shopping for z." The latter pattern matches the input text; the program, using a convention that variables match the longest possible string up to a sentence delimiter (such as a period), assigns z the value, "a new coat," which is given as the answer.</p>	
<p>Comments : This approach is clearly inadequate to answer the kinds of questions people could answer after reading a simple text. Even its ability to answer the most direct questions is delicately dependent on the exact form in which questions are stated and on the variations that were anticipated in the design of the templates and the pattern substitutions that the system uses. In fact, the sheer inadequacy of this program to perform the task may make you wonder how such an approach could even be proposed. This program is substantially farther away from being useful than was the initial program we looked at for tic-tac-toe. Is this just a strawman designed to make some other technique look good in comparison? In a way, yes, but it is worth mentioning that the approach that this program uses, namely matching patterns, performing simple text substitutions, and then forming answers using straightforward combinations of canned text and sentence fragments located by the matcher, is the same approach that is used in one of the most famous "AI" programs ever written- ELIZA, which we discuss in Section 6.4.3. But, as you read the rest of this sequence of programs, it should become clear that what we mean by the term "artificial intelligence" does not include programs such as this except by a substantial stretching of definitions.</p>	

Program 2: This program first converts the input text into a structured internal form that attempts to capture the meaning of the sentences. It also converts questions into that form. It finds answers by matching structured forms against each other.

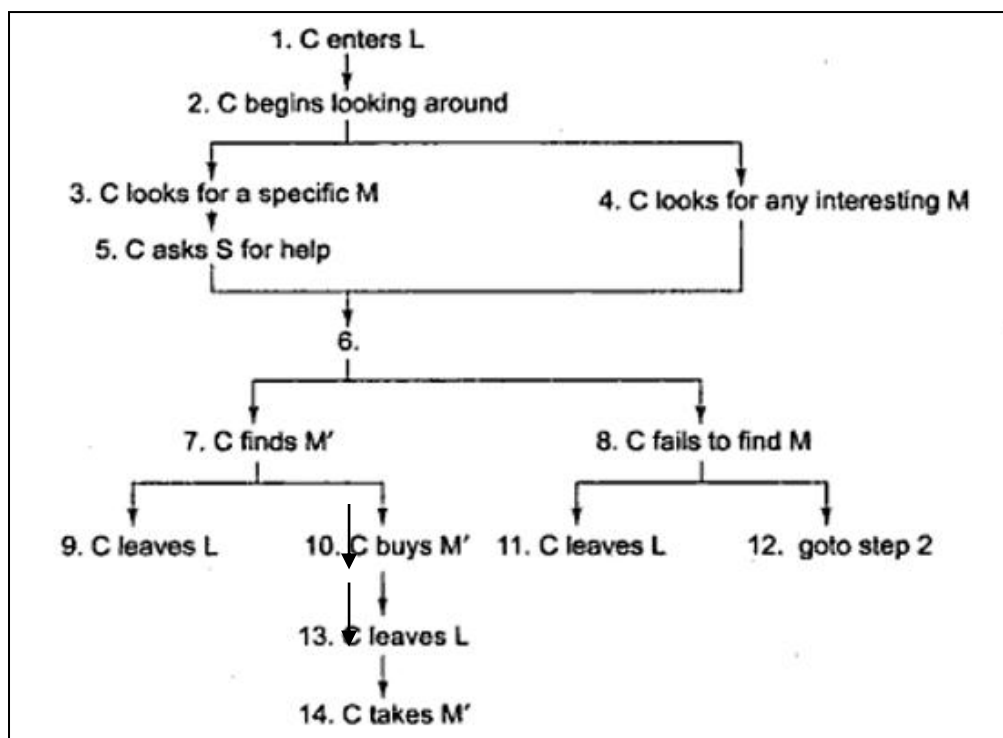
Data Structures	
EnglishKnow	A description of the words, grammar, and appropriate semantic interpretations of a large enough subset of English to account for the input texts that the system will see.
InputText	The input text in character form.
StructuredText	<p>A structured representation of the content of the input text.</p> <p>Processing of Text will generate following slot and filler structure based Semantic Representations:</p> <p style="padding-left: 40px;">Event1</p> <p style="padding-left: 80px;">instance : Finding</p> <p style="padding-left: 80px;">tense: Past</p> <p style="padding-left: 80px;">agent : Mary</p> <p style="padding-left: 80px;">object: Thing 1</p> <p style="padding-left: 40px;">Thing 1</p> <p style="padding-left: 80px;">instance: Coat</p> <p style="padding-left: 80px;">color: Red</p>
Input Question	The input question in character form.
StructQuestion	A structured representation of the content of the user's question.
<p>The Algorithm : Convert the InputText into structured form using the knowledge contained in English Know. Then, to answer a question, do the following:</p> <ol style="list-style-type: none"> 1. Convert the question to structured form, again using the knowledge contained in EnglishKnow. 2. Match this structured form against StructuredText. 3. Return as the answer those parts of the text that match the requested segment of the question. 	
<p>Comments : This approach is substantially more meaning (knowledge)-based than that of the first program and so is more effective. It can answer most questions to which replies are contained in the text, and it is much less brittle than the first program with respect to the exact forms of the text and the questions.</p>	

Program 3:

This program converts the input text into a structured form that contains the meanings of the sentences in the text, and then it combines that form with other structured forms that describe prior knowledge about the objects and situations involved in the text. It answers questions using **this augmented knowledge structure**.

Data Structures	
WorldModel	A structured representation of background world knowledge. This structure contains knowledge about objects, actions, and situations that are described in the input text. This structure is used to construct

	IntegratedText from the input text. For example, Figure 1 shows an example of a structure that represents the system's knowledge about shopping.
EnglishKnow	Same as in Program 2.
InputText	The input text in character form.
IntegratedText	A structured representation of the knowledge contained in the input text (similar to the structured description of Program 2) but combined now with other background, related knowledge.
Input Question	The input question in character form.
StructQuestion	A structured representation of the question.
The Algorithm : To answer question, do the following: 1. Convert the question to structured form as in Program 2 but use WorldModel if necessary to resolve any ambiguities that may arise. 2. Match this structured form against IntegratedText. 3. Return as the answer those parts of the text that match the requested segment of the question.	
Comments : This program is more powerful than either of the first two because it exploits more knowledge. It exploits AI techniques.	



Q9. What are the different class of AI programs?

The programs that perform tasks the way people can do can be divided into two classes:

1. **Programs that do not really fit to the definition of an AI task.** They are the problems that a computer could easily solve. **Example:** Elementary Perceiver and Memorizer (EPAM).
2. Second Class ***of programs that attempts to model human performance*** fall clearly within the definition of AI tasks. They do thinks that are not trivial for the computer.

Q10. At what level of detail, one must try to model human Intelligence?

Q10. What are the reasons to model the human performance using AI programs/tasks?

Answer: The level of detail that one must try to model human intelligence must have genuine reasons such that the modeling must be judicious , legal and rational. Some of the Reasons to model human performance should be:

- a. To test Psychological theories of human performance
- b. To enable computers to understand human reasoning
- c. To enable people to understand computer reasoning
- d. To exploit what knowledge, we can glean from people.

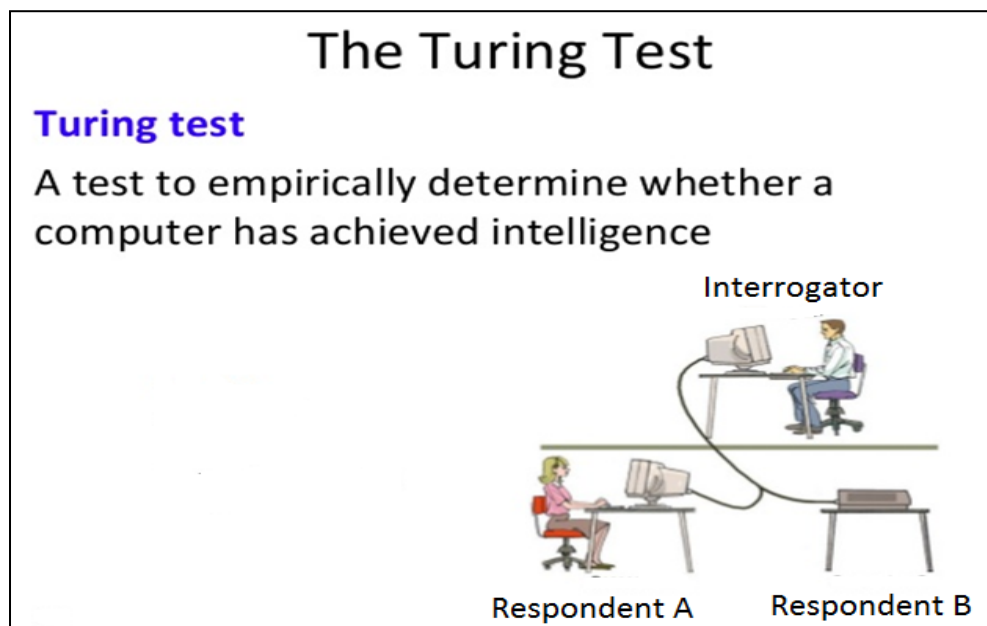
Q11. With a neat diagram explain the Turing Test ?

Q11. How will we know when we have succeeded in building an Intelligent Program?

Answer :

Turing test: Turing in 1950s published an article in the Mind Magazine, which triggered a controversial topic “Can a Machine Think?” In that article he proposed a game named imitation game which was later called TURING TEST. The **Turing test**, is a test of a machine's ability to [exhibit intelligent behavior](#) equivalent to, or indistinguishable from, that of a human. Turing proposed that a human evaluator would [judge natural language conversations](#) between a human and a machine designed to generate human-like responses. The evaluator would be aware that one of the two partners in conversation is a machine, and all participants would be separated from one another. The conversation would be limited to a text-only channel such as a [computer keyboard](#) and [screen](#) so the result would not depend on the machine's ability to render words as speech. If the evaluator cannot reliably tell the machine from the human, the machine is said to have passed the test. The test does not check the ability to give correct answers to questions, only how closely answers resemble those a human would give.

In this game, we need two persons and a computer. One person is interrogator who is in separate room from the computer and the other person. The interrogator asks some questions by typing questions and receives responses the same way. The interrogator has to determine which of them a person is and which of them machine. If machine can make fool of interrogator by not disclosing his actual nature or if interrogator cannot figure out which one of them is machine and which is human then it can be concluded that the machine can think.



Chapter 2. Problems, Problem Space and Search

Q1. What are the four things that we have to do to build an intelligent system to solve the problems?

Answer: To build an intelligent system to solve a particular problem we need to do four things:

1. **Define the problem precisely** (*Precise Specification, Initial Situation, Final Situation and Acceptable Solution*)
2. **Analyze the problem** (Features, Various Possible techniques, etc.)
3. **Isolate and represent the task knowledge** that is necessary to solve the problem.
4. **Choose the best problem-solving techniques and apply** it to the particular problem.

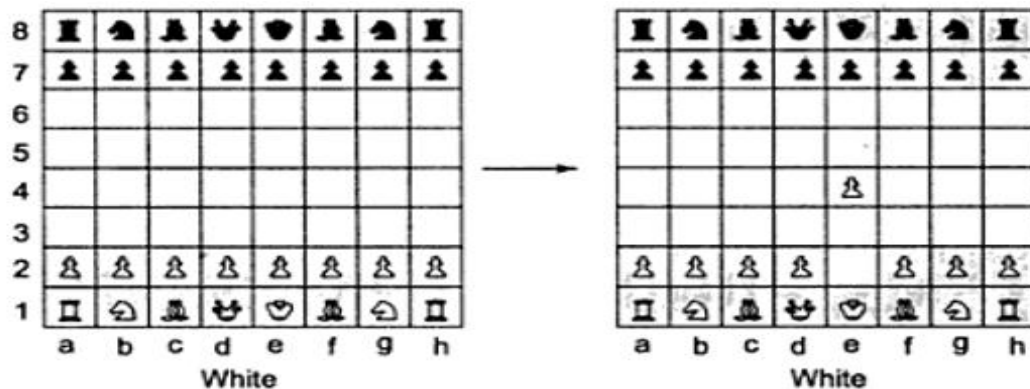
Q2. Explain the two possible ways of representing the Chess Problem.

Answer : In order to describe the problem of playing chess we must take into the consideration the following:

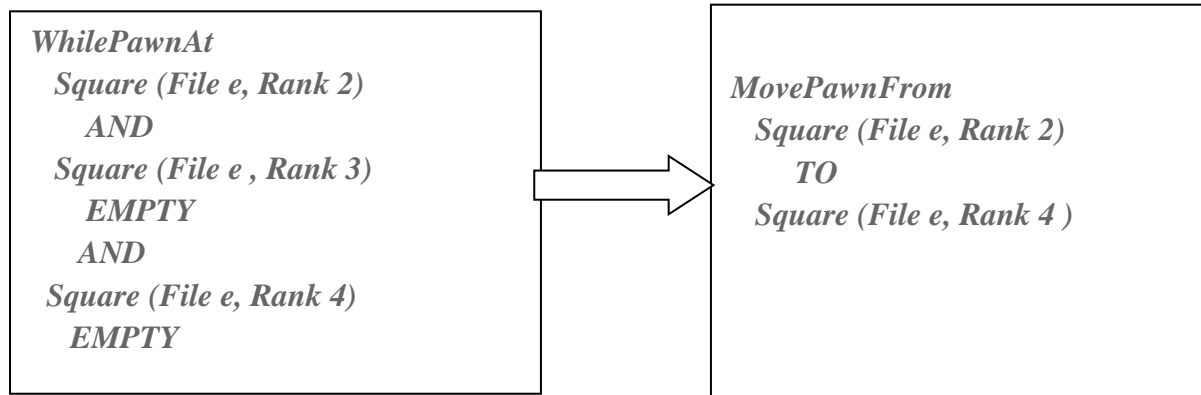
- Specify the starting position of the chess board
- The rules that define the legal moves
- Board positions that represent a win for one side or the other
- Make explicit the previously implicit goal of not only playing a legal game of chess but also winning the games if possible.

Starting position is described as an **8X8 array** where each position contains a symbol standing for the appropriate piece in the official chess opening positions.

One Way to represent the rule is



Another Way to represent the Rule:



Q3. Explain how the AI problems are described formally.

Q3. Explain how the problem is defined as a state space search.

Q3. Give the State Space Representation for the following problems

a. Tic Tac Toe

b. Water Jug Problem

c. Travelling Sales Person Problem

Answer:

Formal Description of a Problem: In order to provide a formal description of a problem we must do the following:

- a. Define a state space that contains all possible configuration of the relevant objects
- b. Define the initial States
- c. Define the Goal /Final States
- d. Specify the Rules that describe the action (operators) available.

Examples:**1. The Tic Tac Toe as a State Space Search Representation****a. Configuration of the Board (State Space)**

- Board is Vector of 9 squares of 3 X 3 Matrix Form
- Next Move of a Players: X or O
- Board Configuration may change accordingly to each move of the player

b. Initial State: Board is Empty by X's Turn**c. Final States:** Three X's in Row / Three O's in Row / All Cells Full**d. Rules:**

- At a time one square can be filled in any direction
- Initial turn is of X
- Out of X and O players who will form the row First will be the winner
- If No Row is formed but the board is filled the Match is draw.

2. A Water Jug Problem: You are given two jugs a 4 Gallon Jug one and 3 Gallon Jug one. Neither has any measuring marker on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 Gallon of Water into the 4 Gallon Jug?**Solution:****a. State Space:** The state space for this problem can be described as the set of ordered pairs of integers (x, y). Where

- **x** represents the number of Gallons of water in the 4 Gallon Jug, such that $x = 0, 1, 2, 3$ or 4

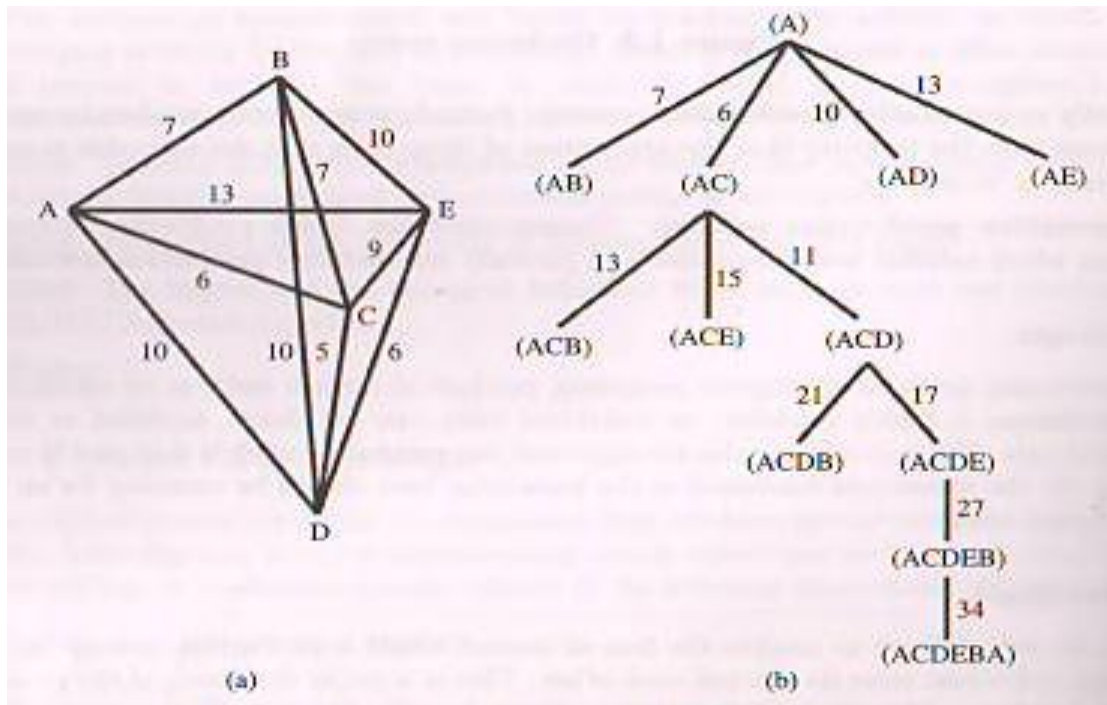
- **y** represents the quantity of water in the 3 Gallons jug, such that $y = 0, 1, 2$ or 3 .
- b. **Initial State:** The Start State is $(0,0)$
- c. **Goal State:** The Final State is $(2,n)$
- d. **Rules:** The Production rule for the Water Jug Problem is given below:

Rule #	Rule	Descriptions
1	$(x,y) \text{ if } x < 4 \rightarrow (4,y)$	Fill the 4 Gallon Jug
2	$(x,y) \text{ if } y < 3 \rightarrow (x,3)$	Fill the 3 Gallon Jug
3	$(x,y) \text{ if } x > 0 \rightarrow (x-d,y)$	Pour Some Water out of 4 Gallon Jug
4	$(x,y) \text{ if } y > 0 \rightarrow (x,y-d)$	Pour Some Water out of 3 Gallon Jug
5	$(x,y) \text{ if } x > 0 \rightarrow (0,y)$	Empty the 4 Gallon Jug on the Ground
6	$(x,y) \text{ if } y > 0 \rightarrow (x,0)$	Empty the 3 Gallon Jug on the Ground
7	$(x,y) \text{ if } x+y \geq 4 \text{ and } y > 0 \rightarrow (4,(y-(4-x)))$	Pour the water from the 3 Gallon Jug into the 4 Gallon Jug until the 4 Gallon Jug full
8	$(x,y) \text{ if } x+y \geq 3 \text{ and } x > 0 \rightarrow (x-(3-y),3)$	Pour water from the 4 Gallon jug into the 3 Gallon jug until the 3 Gallon jug is full
9	$(x,y) \text{ if } x+y \leq 4 \text{ and } y > 0 \rightarrow (x+y,0)$	Pour all the water from the water from the 3 Gallon jug into the 4 Gallon jug.
10	$(x,y) \text{ if } x+y \leq 3 \text{ and } x > 0 \rightarrow (0,x+y)$	Pour all the water from the 4 G jug into the 3 G jug
11	$(0,2) \rightarrow (2,0)$	Pour the 2 Gallon jug from the 3 Gallon jug into the 4 Gallon Jug
12	$(2,y) \rightarrow (0,y)$	Empty the 2 Gallon jug in the 4 Gallon Jug on the Ground.

One Solution to the Problem

Gallons in the 4 Gallon Jug	Gallons in the 3 Gallon Jug	Rule Applied
0	0	Initial State
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 or 12
2	0	9 or 11

3. Travelling Salesman Problem: Enumerate the Global database for the following TS problem and find the solution. A Salesman must visit each of five cities (A,B,C,D and E) as shown in the Figure a.



The goal is to find the minimum path. The rule is to branch out from a node which has minimal distance from the root node.

Answer:

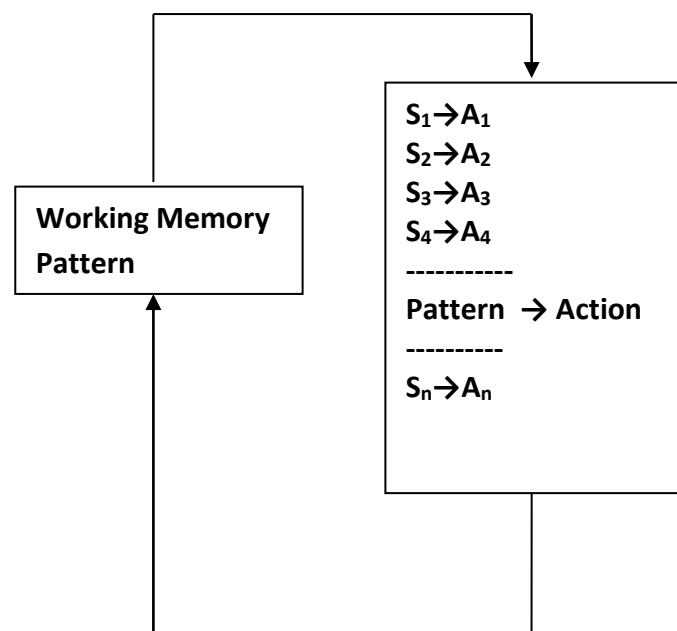
- State Space:** All possible tours.
- Initial State:** Any City say A in the above example.
- Goal State:** Finding the **Shortest tour** connecting all cities.
- Rules:** The Salesman must visit all the given cities once and return to the original city.

Solution: Let A be selected as the starting node. We can specify the global database in the form of a graph. The minimal path is A->C->D->E->B->A and cost = 34. Figure b shows the global database for the TSP.

Q4. With a neat diagram Explain what is Production System. Describe all the major components of AI production system.

Answer : A production system (or production rule system) is a computer program typically used to provide some form of artificial intelligence, which consists primarily of a set of rules about behavior. These rules, **termed productions**, are a basic representation found useful in automated planning, expert systems and action selection. A production system provides the mechanism necessary to execute productions in order to achieve some goal for the system.

Productions consist of two parts: A *sensory precondition* (or "IF" statement) and an **action** (or "THEN"). If a production's precondition matches the current state of the world, then the production is said to be triggered. If a production's action is executed, it is said to have fired. A production system also contains a database, sometimes called working memory, which maintains data about current state or knowledge, and a rule interpreter. The rule interpreter must provide a mechanism for prioritizing productions when more than one is triggered. A production system control loop until working memory pattern no longer matches the conditions of any production.



A production system consists of rules and factors. Knowledge is encoded in a declarative form which comprises of a set of rules of the form:

Situation \rightarrow Action

The major components of an AI production system are

i. A Global database: The global database is the *central data structure* used by an AI production system. It consists of one or more knowledge databases that contain whatever information is appropriate for the particular task. Some parts of the database may be permanent, while others may be temporary and only exist during the solution of the current problem. The information in the databases may be structured in any appropriate manner.

ii. A set of production rules: A set of production rules, which are of the form **A \rightarrow B**. Each rule consists of left hand side constituent that represent the current problem state and a right-hand side that represent an output state. A rule is applicable if its left-hand side matches with the current problem state. A set of rules of the form **C_i \rightarrow A_i** where **C_i** is the condition part and **A_i** is the action part. The condition determines when a given rule is applied, and the action determines what happens when it is applied.

iii. A control system/Strategy: A control strategy that determines the order in which the rules are applied to the database, and provides a way of resolving any conflicts that can arise when several rules match at once. The control system chooses which applicable rule should be applied and ceases computation when a termination condition on the database is satisfied. If several rules are to fire at the same time, the control system resolves the conflicts. The control strategy specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.

iv. Rule Applier: A rule applier which is the computational system that implements the control strategy and applies the rules. A rule applier, which checks the capability of rule by matching the content state with the left-hand side of the rule and finds the appropriate rule from database of rules. The important roles played by production systems include a powerful knowledge representation scheme. A production system not only represents knowledge but also action. It

acts as a bridge between AI and expert systems. Production system provides a language in which the representation of expert knowledge is very natural. We can represent knowledge in a production system as a set of rules of the form

If (condition) THEN (condition)

along with a control system and a database. The control system serves as a rule interpreter and sequencer. The database acts as a context buffer, which records the conditions evaluated by the rules and information on which the rules act. The production rules are also known as condition – action, antecedent – consequent, pattern – action, situation – response, feedback – result pairs.

For example,

**If (you have an exam tomorrow)
THEN (study the whole night)**

The production system can be classified as monotonic, non-monotonic, partially commutative and commutative.

Q5. Describe all the four classes of Production Systems.

Answer:

The Four classes of production systems are :

- 1. A monotonic production system:** A system in which the application of a rule never prevents the later application of another rule that could have also been applied at the time the first rule was selected.
- 2. A non-monotonic production system:** Here the application of rule prevents later application of another rule which may not have been applied at the time the first rule was selected i.e. it is a system in which the above rule is not true i.e, the monotonic production system rule not true.
- 3. A partially commutative production system:** A production system in which the application of a particular sequence of rules transforms state ***X into state Y***, then any permutation of those rules that is allowable also transforms state ***X into state Y***.
- 4. A commutative production system:** Is a program which satisfies both monotonic and partially commutative production System.

Figure below illustrates the relationship between different types of production systems with example.

	Monotonic	Non-Monotonic
Partially Commutative	Theorem Proving	Robot Navigation
Not Partially Commutative	Chemical Synthesis	Bridge

Q6. What are the advantages and disadvantages of Production System.

Answer:

Advantages of production systems: -

1. Production systems provide an excellent tool for structuring AI programs.
2. Production Systems are highly modular because the individual rules can be added, removed or modified independently.
3. The production rules are expressed in a natural form, so the statements contained in the knowledge base should be the recording of an expert thinking out loud.

Disadvantages of Production Systems: -

1. One important disadvantage is the fact that it may be very difficult analyzes the flow of control within a production system because the individual rules don't call each other.
2. Several Production systems exist for several types of problems. The efficiency varies from problem to problem.

Q7. What are the different types of search techniques used in Artificial Intelligence. Give Examples.

Q7. Give the classification of different types of search techniques

Answer: The basic search algorithms/techniques used in Artificial Intelligence can be classified as follows:

- **Uninformed(Blind) search:** Breadth-first, Depth-first search, Depth limited search, Iterative deepening and Bidirectional search.

- **Informed (Heuristic) search:** Search is guided by an evaluation function: Generate and Test, Hill Climbing, Best First Search, Problem Reduction, Constraint Satisfaction and Means End Analysis.

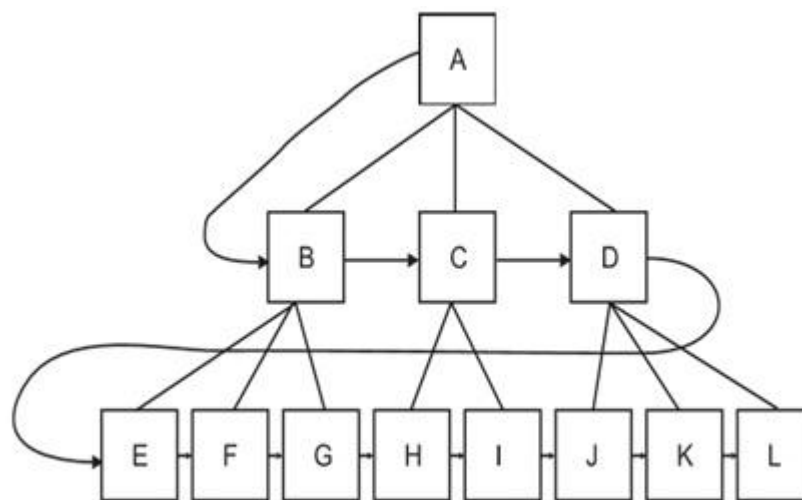
Q8. Explain the following uninformed search techniques (Algorithm) with example

1. Breadth First Search and BFS Algorithm

2. Depth First Search and DFS Algorithm

Answer:

1. Breadth First Search: Here the nodes are expanded one level at a time. 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., the nodes will be explored in order A, B, C, D, E, F, G, H, I, J, K, L:

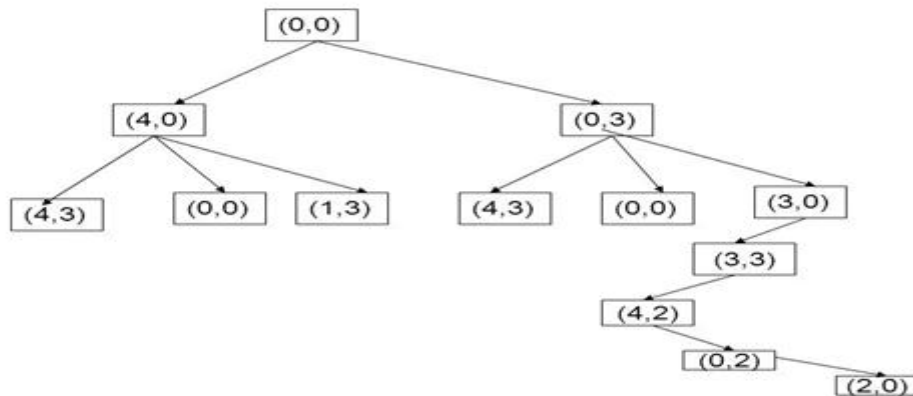


Breadth-First Search Tree

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).

Example: BFS Tree for Water Jug Problem



Breadth first search Algorithm

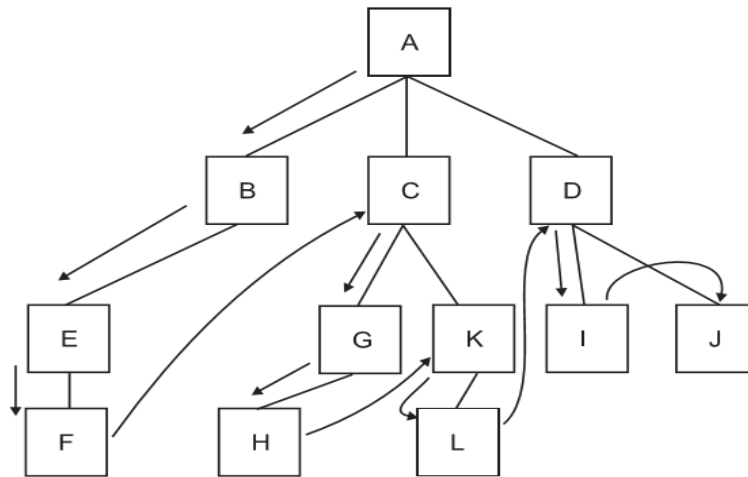
S1. Create a variable called NODE_LIST and set it to the initial state

S2: Until a goal state is found or NODE_LIST is empty

- a. Remove the First Element from NODE_LIST and Call it E
 - I. 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 the new state is a goal state, quit and return this state
 - III. Otherwise add the new state to end of NODE_LIST

2. Depth First Search: (*Expand one of the nodes at the deepest level*). 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 below:

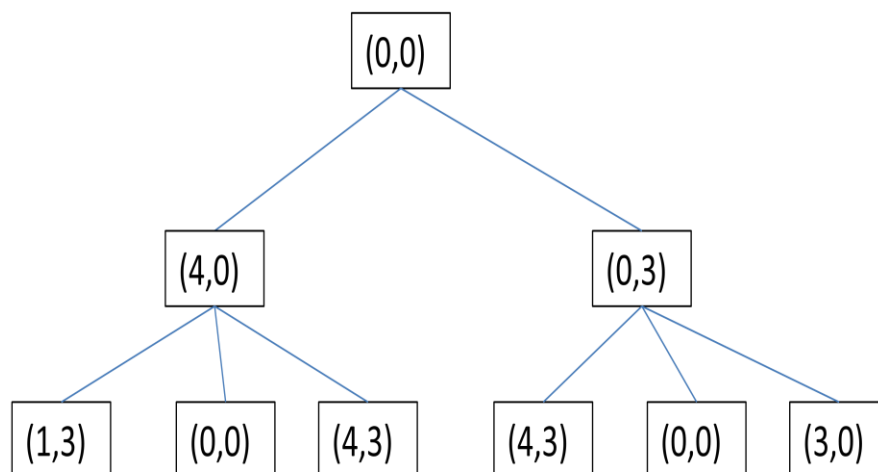


The DFS has lesser space complexity, because at a time it needs to store only single path from root to leaf node. The data structure used for DFS is Last-In First-Out (LIFO).

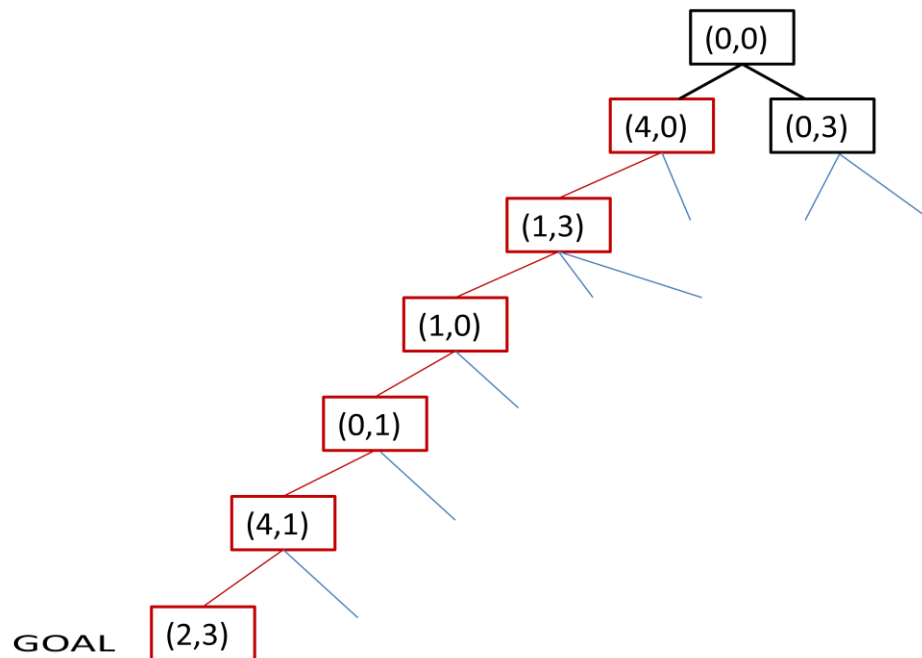
Depth first search Algorithm

1. If the initial state is a goal state quit and return success
2. Otherwise do the following until success or failure is signaled
 - a. Generate a successor E of initial State. If there are no successor signal failure.
 - b. Call DFS with E as the initial state
 - c. If success is returned , signal success , otherwise continue in this loop

Water Jug Problem Solution using DFS

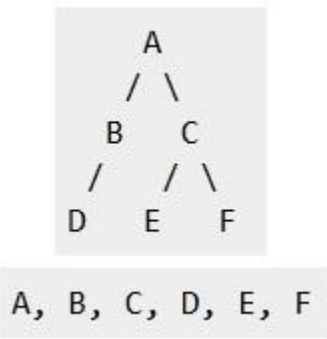
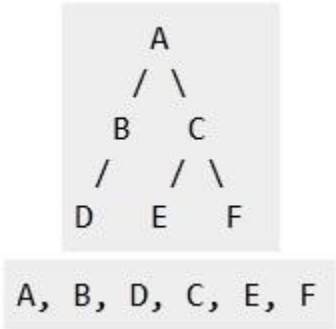


One of the solution tree is



Q9. Differentiate between DFS and BFS. List the advantages and disadvantages of DFS and BFS.

BFS	DFS
BFS Stands for “ Breadth First Search ”.	DFS stands for “ Depth First Search ”.
BFS starts traversal from the root node and then explore the search in the level by level manner i.e. as close as possible from the root node.	DFS starts the traversal from the root node and explore the search as far as possible from the root node i.e. depth wise.
Breadth First Search can be done with the help of queue i.e. FIFO implementation. This algorithm works in single stage. The visited vertices are removed from the queue and then displayed at once.	Depth First Search can be done with the help of Stack i.e. LIFO implementations. This algorithm works in two stages – in the first stage the visited vertices are pushed onto the stack and later on when there is no vertex further to visit those are popped-off.

<p>BFS is slower than DFS.</p> <p>BFS requires more memory compare to DFS.</p> <p>Applications of BFS</p> <ul style="list-style-type: none"> > To find Shortest path > Single Source & All pairs shortest paths > In Spanning tree > In Connectivity 	<p>DFS is more faster than BFS.</p> <p>DFS require less memory compare to BFS.</p> <p>Applications of DFS</p> <ul style="list-style-type: none"> > Useful in Cycle detection > In Connectivity testing > Finding a path between V and W in the graph. > useful in finding spanning trees & forest.
<p>BFS is useful in finding shortest path. BFS can be used to find the shortest distance between some starting node and the remaining nodes of the graph.</p> <p>Example:</p> 	<p>DFS is not so useful in finding shortest path. It is used to perform a traversal of a general graph and the idea of DFS is to make a path as long as possible, and then go back (backtrack) to add branches also as long as possible.</p> <p>Example:</p> 

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 DFS may follow single unfruitful path for a very long time
- 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. In contrast DFS may find a long path to a solution in one part of the tree , when a shorter path exists in some other , unexplored part of the tree.

Advantages of Depth First Search (DFS)

1. DFS requires less memory space since only the nodes on the current path are stored. In BFS all the tree that has so far being generated must be stored
2. If care is taken in ordering the alternative successor states DFS may find a solution without examining much of the search space at all. In BFS all parts of the tree must be examined to level n before any nodes level $n+1$ can be examined.

Q10. Give the solution to water jug Problem using DFS and BFS.

Answer: Refer Q.No 8

Q11. What are the Issues in the Design of search programs? Explain.

Answer: The Issues in the Design of search programs are as follows

1. The direction in which to conduct the search (forward vs backward reasoning)
2. How to select applicable rules (matching)
3. How to represent each node of the search process (knowledge representation problem and frame problem)
4. Search tree versus Graph

Explanation: The students are expected to write the explanation based on their domain knowledge.

Chapter 3: Heuristic Search Techniques

Q1. Explain what is Heuristic. And Give examples for the problems where heuristic is applied.

Answer: A heuristic technique (*meaning : "find" or "discover" or "Guide"*), often called simply a heuristic, is any approach to problem solving, learning, or discovery that employs a practical method not guaranteed to be optimal or perfect, **but sufficient for the immediate goals.**

The introduction of 'Heuristics' can improve the efficiency of the search process, probably sacrificing the completeness. The dictionary meaning of term heuristic is GUIDE. Heuristic works like a tour guide. **E.g.** imagine that you are lost in a jungle and you are searching for road which can lead you to a particular city. A tour guide can guide you by removing so many unwanted roads by providing you important information so that you can reach your destination more quickly.

Where finding an optimal solution is impossible or impractical, heuristic methods can be used to speed up the process of finding a satisfactory solution. Heuristics however do not guarantee any completeness. All that can be said about heuristic is that it offers solutions which are good enough most of the time

Following are the problems where heuristics are applied

- 1) Problems for which no exact algorithms are known and one needs to find an approximation and satisfying solution for example: Speech recognition, and computer vision etc.
- 2) Problems for which exact solutions are known, but computationally they are infeasible. E.g. chess and cube game

Q2. Explain what is heuristic search with example.

Answer:

Heuristic Search : In un-informed search the search procedure have no other information except initial node, goal node and set of legal moves or rules where **as in heuristic search** along with this information other domain specific

information is also available like *distance of the current node and the goal node or the cost of moving from current node to goal node*, which can help the search procedure to find the goal state more quickly and efficiently that is why this type of search is also known **as informed search**.

Example: One example of a good general-purpose heuristic is nearest neighbor heuristic applied to the Travelling Salesman Problem:

1. Arbitrarily select a starting city
2. To select the next city, look at all cities not yet visited and select the one closest to the current city. Go to it next
3. Repeat Step 2 until all cities have been visited

Q3. Explain what are heuristic functions with examples.

Heuristic Functions: The heuristics which are required for solving problems are generally represented as heuristic functions. Heuristic functions convert the problems states in to quantitative form. The purpose of this is to guide the search procedure in most profitable direction by suggesting which path has to be followed first when more than one path are available. Heuristic function can be the distance between current state and the goal state or the distance between two consecutive states. Hamming distance is a heuristic function used in 8 x 8 tile puzzle and it gives the indication of how many tiles are in the position what they are supposed to be.

- $h(n)$, which takes a node n and returns a non-negative real number that is an estimate of the path cost from node n to a goal node.
- The heuristic function is a way to inform the search about the direction to a goal. It provides an informed way to guess which neighbor of a node will lead to a goal.

Some Simple Heuristic Functions are listed below:

- **Chess:** The material advantage of our side over the opponent
- **Travelling Salesman Program:** To sum of the distances travelled so far
- **Tic Tac Toe:**
 - 1 for each row in which we could win and in which we already have one piece
 - 2 for each such row in which we have two pieces

Q4. Give examples for the search algorithms which use the heuristic functions.

Answer: Following are the search algorithms which use the heuristic functions:

1. Generate and Test
2. Hill Climbing
3. Best First Search
4. Problem Reduction
5. Constraint Satisfaction
6. Mean End Analysis

Q5. With examples explain all seven problem characteristics.

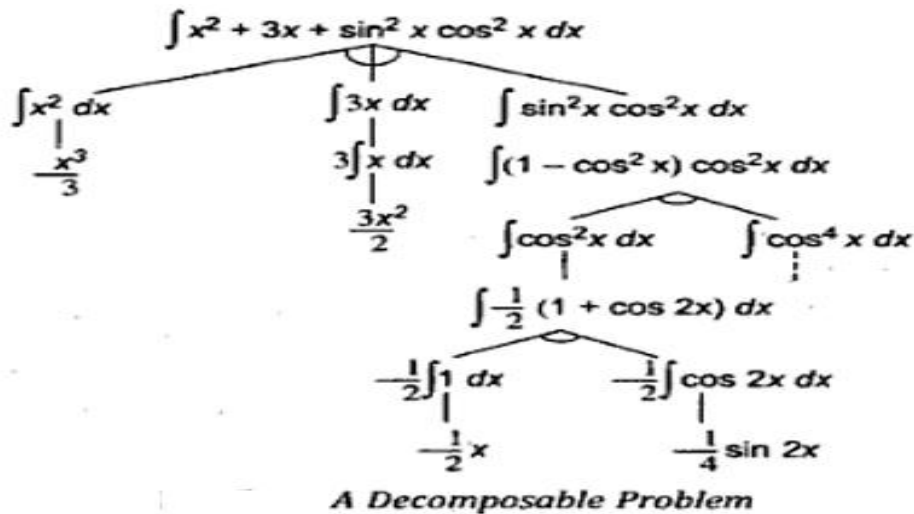
Answer: The (Seven) Problem Characteristics are :

1. Is the problem decomposable?
2. Can solution steps be ignored or undone?
3. Is the universe predictable?
4. Is a good solution absolute or relative?
5. Is the solution a state or a path?
6. What is the role of knowledge?
7. Does the task require human-interaction?

1. Is the problem decomposable?

The problem is decomposable if the problem can be break down into smaller problem each of which we can then solve by using a smaller collection of specific rules.

Example 1 :



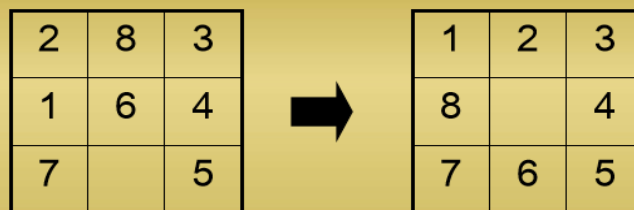
2. Can solution steps be ignored or undone?

In mathematical theorem proving the solution steps can be ignored. In bridge or chess the solution steps are not ignorable. Also they are not recoverable. But in 8 puzzle problem the solution steps are recoverable.

- **Classes of problems**

- **Ignorable** (Eg: Theorem Proving) in which solution steps can be ignored
- **Recoverable** (Eg: 8 Puzzle) in which solution steps can be undone
- **Irrecoverable** (Eg: Chess) in which solution steps cannot be undone.

The 8-Puzzle



Moves can be undone and backtracked.

3. Is the universe predictable?

- **Predictable:** In 8 puzzle every time we make a move we know exactly what will happen, this means that it is possible to plan an entire sequence of moves and be confident that we know what the resulting state will be. (The Universe Outcome is Certain)
- **Un Predictable :** In Playing Bridge , We Cannot know exactly where all the cards are or what the other players will do on their turns.

4. Is the solution absolute or relative?

Consider the problem of answering question based on a database of simple facts, such as the following:

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 Pompeian's died when the volcano erupted in 79 A.D.
6. No mortal lives longer than 150 years.
7. It is now 1991 A.D.

Suppose we ask a question " Is Marcus alive? " By representing each of these facts in a formal language such as predicate logic and then using formal inference methods we can fairly easily derive and answer to the question.

Two ways to decide the answer

Way1:

1	Marcus was a man	1
4	All men are mortal.	4
8	Marcus is a mortal	1,4
3	Marcus was born in 40 A.D.	3
7	It is now 1991 A.D.	7
9	Marcus age is 1951 years	3,7
6	No mortal lives longer than 150 years	6
10	Marcus is dead	8,6,9

Way2:

7	It is now 1991 A.D.	7
5	All Mortal's died when the volcano erupted in 79 A.D.	5
11	All Mortal's are dead now	7,5
2	Marcas was Pompeian	2
12	Marcus is dead	11,2

In above both cases the solution is **relative** to the other.

5. Is the solution is State or Path ?

For the problem of the natural language understanding , ***the solution is a state of the world***. Consider the problem of finding a consistent interpretation for the sentence “The **bank president ate a dish of pasta salad with the fork** “ . There are several components of this sentence each of which in isolation may have more than one interpretation. But the component must form a ***coherent whole*** and so they constrain each other interpretation.

- “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?

-

Contrast to this with ***the water jug problem***, what we really must report is not the final state but the path that we found to that state. Thus a statement of solution to this problem must be a sequence of operation that produces the final state. This problem solution is ***path to the state***.

6. What is the role of knowledge?

Is a large amount of knowledge absolutely required to solve the problem or is knowledge important only to constrain the search? In 8 puzzle or Chess game

we get a solution with the help of knowledge base. (Rules for determining the legal moves).

But when we consider the problem of scanning daily newspaper to decide which political party will win in upcoming election, it would have to know such things as

- The name of the candidate of each party
- Major issues/achievement to support party
- Major issues to oppose the party
- And so on –

These two problems Chess and Newspaper story understanding, illustrate the difference between problems for which a lot of knowledge is important only to constrain the search for a solution and those for which a lot of knowledge is important only to constrain the search for a solution and those for which a lot of knowledge is required even to be able to recognize a solution.

Q6. Write a brief note on Problem Classification.

Answer:

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. Broad classes to which the problems falls is associated with a generic control strategy that is appropriate for solving the problems.

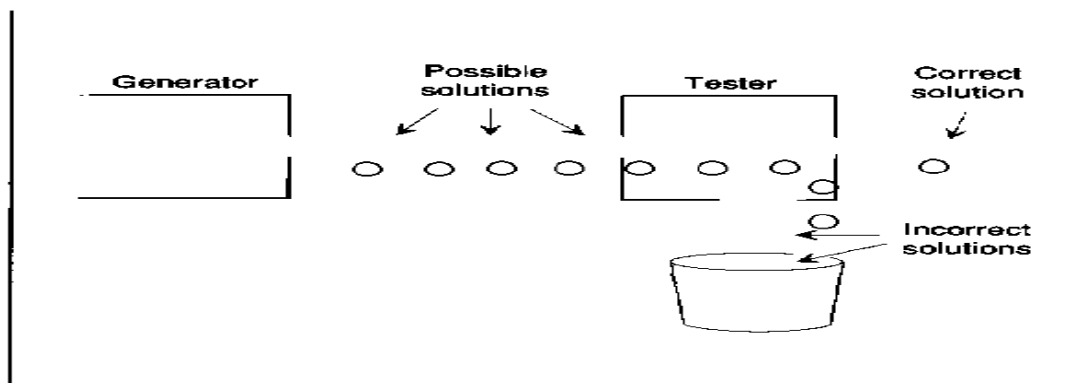
1. **Classification:** Medical diagnosis tasks, Diagnosis of fault in mechanical devices.
2. **Propose and Refine:** Design and Planning Problems

Q7. Explain the following Heuristic Search techniques with algorithms and examples

1. Generate and Test
2. Hill Climbing
3. Best First Search and A* Search
4. Problem Reduction and AO* Search
5. Constraint Satisfaction
6. Means End Analysis

Answer:

1. **Generate-and-Test:** Here the possible solution is generated and *the generated solution is tested* by comparing to the set of acceptable goal states. The correct solution will be considered and incorrect solutions will be rejected. The working of Generate and test solution is illustrated in the figure below:



Generate and Test Algorithm

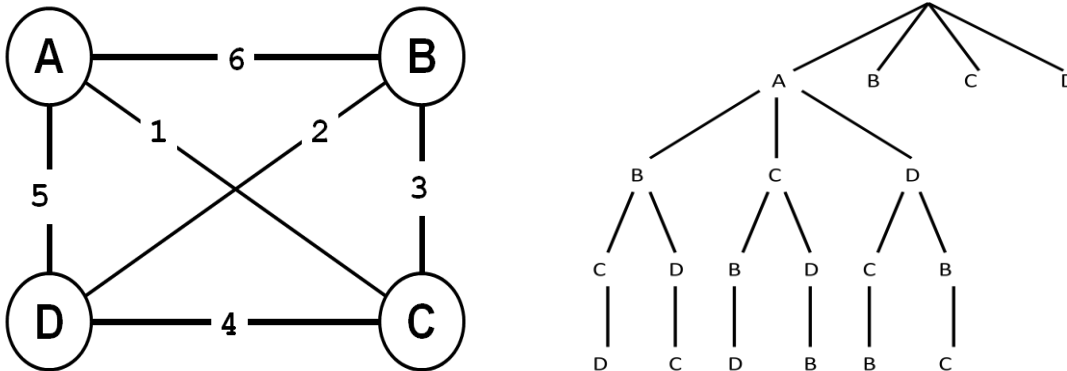
1. Generate a possible solution.
2. Test to see if this is actually a solution by comparing to the set of acceptable goal states.
3. If a solution has been found, quit.
Otherwise, return to step 1.

Generate and Test Solution may be a *systematic form or random form*. In systematic form, it is simply an exhaustive search of the problem space. The

random form is also known as the **British Museum** method a reference to a method for finding an object in the British Museum by wandering randomly.

Example - Traveling Salesman Problem (TSP)

Here Traveler needs to visit n cities. The distance between each pair of cities are given. The goal is to determine the shortest route that visits all the cities once. If number of cities is $n=80$ it will take millions of years to solve exhaustively!



For the problem of 4 cities given above, **generation of possible solutions** is done in lexicographical order of cities as given below:

1. A - B - C - D
2. A - B - D - C
3. A - C - B - D
4. A - C - D - B
- ...

2.Hill Climbing : Hill Climbing is a variant of generate and test in **which feedback** from the test procedure is used to help the generator decide which direction to move in the search space. Following are the different types of Hill Climbing algorithm.

1. Simple Hill Climbing
2. Steepest Ascent Hill Climbing
3. Simulated Healing

2.1 Simple Hill Climbing: It uses heuristic to move only to states that are better than the current state. Always moves to better state when possible. The process ends when all operators have been applied and none of the resulting states are better than the current state. The simplest way to implement hill climbing is as follows.

Algorithm: Simple Hill Climbing (Local Search)

1. **Evaluate the initial state.** If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. **Loop until a solution is found or until there are no new operators left to be applied in the current state:**
 - (a) Select an operator that has not yet been applied to the current state and apply it to produce a new state.
 - (b) Evaluate the new state.
 - (i) If it is a goal state, then return it and quit.
 - (ii) If it is not a goal state but it is better than the current state, then make it the current state.
 - (iii) If it is not better than the current state, then continue in the loop.

2.2 Steepest-Ascent Hill Climbing (Gradient Search): It is a variation on simple hill climbing. Instead of moving to the *first* state that is *better*, move to the best possible state that is **one move away**. The order of operators does not matter. Not just climbing to a better state, climbing up the *steepest* slope is better. The algorithm for steepest hill climbing is as given below :

Algorithm: Steepest-Ascent Hill Climbing

1. **Evaluate the initial state.** If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. **Loop until a solution is found or until a complete iteration produces no change to current state:**
 - (a) Let *SUCCESSOR* (*S*) be a state such that any possible successor of the current state (CS) will be better than *S*.
 - (b) For each operator that applies to the current state do:
 - (i) Apply the operator and generate a new state (NS).
 - (ii) Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to *S*. If it is better, then set *S* to this state. If it is not better, leave *S* alone.
 - (c) If the *S* is better than CS, then set CS to *S*.

- 2.3 Simulated Annealing:** The algorithm is derived based on the behavioural patterns of metals that changes their state when it is heated to certain temperature. The algorithm is patterned after the physical process of annealing, known as **Physical annealing** in which physical substances are melted and then gradually cooled until some solid state is reached. The goal is to produce a minimal-energy state. The rate at which the system is cooled is called the **annealing schedule**. If the temperature is lowered sufficiently slowly, then the goal will be attained.

The probability for a transition to a higher energy state is given by the formula : $e^{-\Delta E/kT}$

Where ΔE is the positive change in the energy level, T is the temperature and k is the Boltzmann's constant.

The algorithm is described below:

Simulated Annealing Algorithm

1. **Evaluate the initial state.** If it is a goal state, then return it and quit. Otherwise set initial state as the current state.
2. Initialize **BEST SO FAR** to the current state.
3. **Initialize T** according to the annealing schedule
4. **Loop until a solution is found or there are no new operators left to be applied:**
 - a. Select and applies a new operator which has not yet applied.
 - b. **Evaluate the new state:**
 - If the **new state is goal state**, then return it and quit.
 - If it is **not a goal state and better than the current state** then make it as the current state. Also set BEST-So- FAR to this new state.
 - If it is **not better than the current state**, then
Compute $\Delta E = \text{Val}(\text{current state}) - \text{Val}(\text{new state})$ make it the current state with probability $P = e^{-\Delta E/kT}$
 - c. **Revise T** as necessary according to the annealing schedule
5. Return **BEST –So-FAR** as the answer.

Note : The algorithm for simulated annealing is only slightly different from the simple hill climbing procedure . The three differences are :

- The annealing schedule must be maintained
- Moves to worse states may be accepted
- It is a good idea to maintain, in addition to the current state the best state found so far.


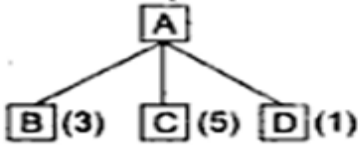
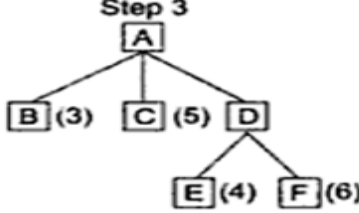
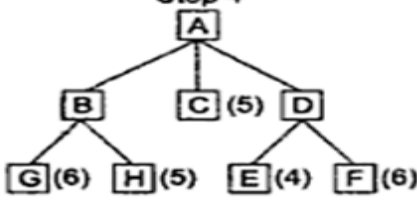
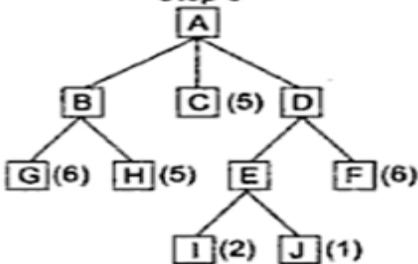
3. Best First Search:

Depth-first search is good because it allows a solution to be found without all competing branches having to be expanded. **Breadth-first search** is good because it does not get trapped on dead-end paths. **One way of combining the two is to follow** a single path at a time, but switch paths whenever some competing path looks more promising than the current one does. At each step of the **best-first search process**, we select the *most promising of the nodes* we have generated so far. This is done by applying *an appropriate heuristic function* to each of them. We then expand the chosen node by using the rules to generate its successors. If one of them is a solution, we can quit. If not, all those new nodes are added *to the set of nodes generated so far*. Again, the most promising node is selected and the process continues. The Best First Search Algorithm is given below:

Best First Search Algorithm

1. Start with *OPEN* containing just the initial state.
2. Until a goal is found or there are no nodes left on *OPEN* do:
 - (a) Pick the best node on *OPEN*.
 - (b) Generate its successors.
 - (c) For each successor do:
 - (i) If it has not been generated before, evaluate it, add it to *OPEN*, and record its parent.
 - (ii) If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already, have.

Example: The Figure below illustrates the example for Best First Search using **OR Graphs**.

Best First Search	
Step	Priority Queue Status
Step 1 	OPEN = { A } CLOSE = { }
Step 2 	OPEN = { A,D } CLOSE = { B,C }
Step 3 	OPEN = { A,D,E } OR OPEN = { A,B } CLOSE = { C }
Step 4 	OPEN = { A,D,E } CLOSE = { B,C,G,H }
Step 5 	OPEN = { A,D,E,J } CLOSE = { B,C,G,H,F,I }

3.1 A* Search Algorithm:

A* (A star) is the most widely known form of **Best-First search**. It evaluates nodes by combining $g(n)$ and $h(n)$.

That is by computing the evaluating function

$$f(n) = g(n) + h(n) .$$

– Where

- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

The complete A* algorithm, its advantages and disadvantages is given below:

A* Algorithm

1. Start with *OPEN* holding the initial nodes.
2. Pick the *BEST* node on *OPEN* such that $f(n) = g(n) + h(n)$ is minimal.
3. If *BEST* is goal node quit and return the path from initial to *BEST* **Otherwise**
4. Remove *BEST* from *OPEN* and all of *BEST*'s children, labeling each with its path from initial node.

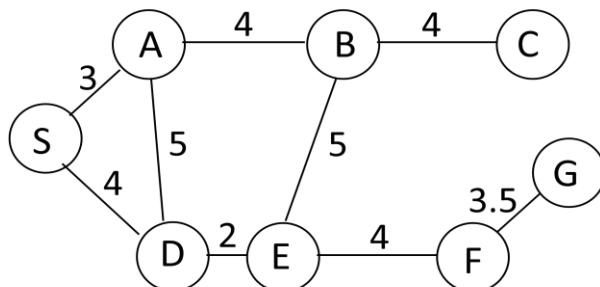
Advantages

- It is complete and optimal.
- It is the best one from other techniques.
- It is used to solve very complex problems.
- It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A*.

Disadvantages

- This algorithm is complete if the branching factor is finite and every action has fixed cost.
- The speed execution of A* search is highly dependent on the accuracy of the heuristic algorithm that is used to compute $h(n)$.
- It has complexity problems.

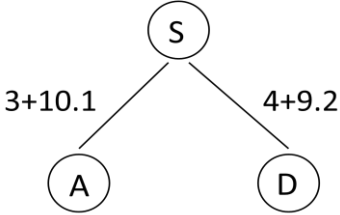
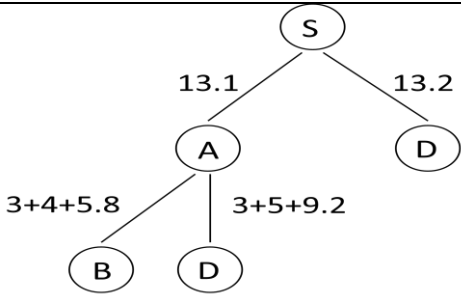
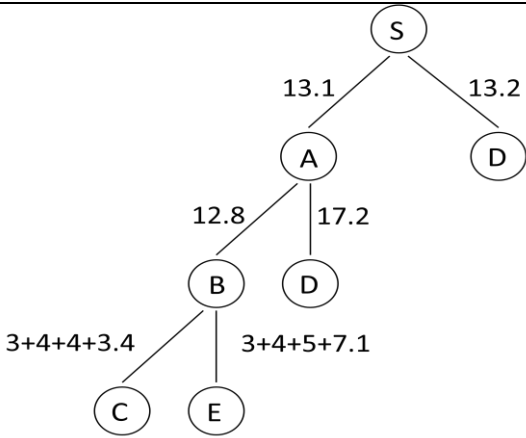
Example for A* Search Techniques: Consider the graph of different cities like S, A, B, C, D, E, F and G. The objective is to find the optimal path from source S to Goal City G.

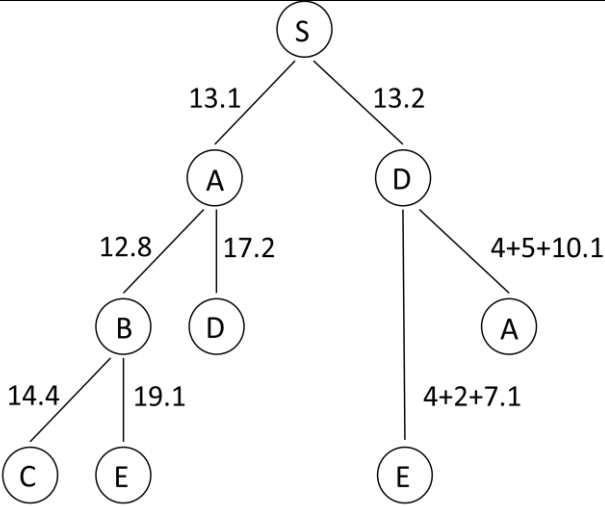
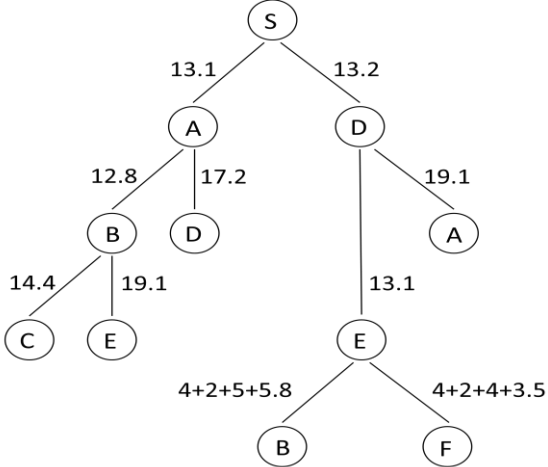
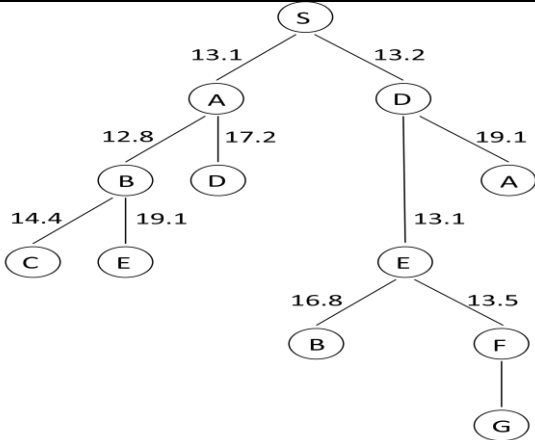


The Straight line distance, i.e., heuristic value $h(n)$, from each city to goal city is given in table .

City	Straight Line Distance to Goal City ($h(n)$)
S	11.5
A	10.1
B	5.8
C	3.4
D	9.2
E	7.1
F	3.5
G	0

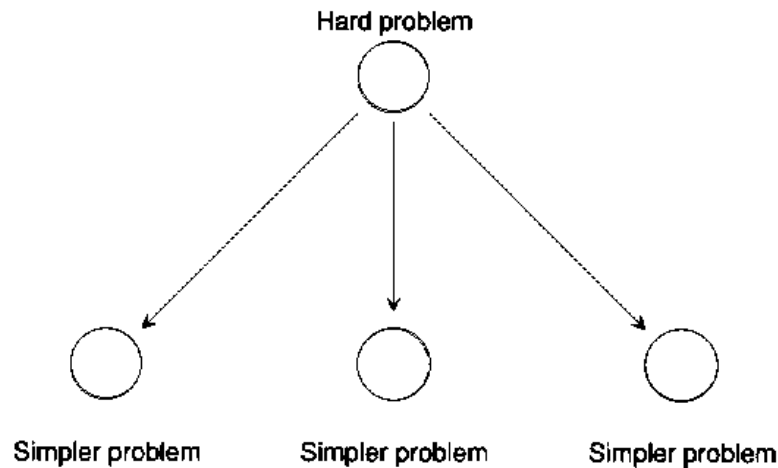
The simulation of each steps of A* algorithm to find the goal is given below:

A*, step 1	 <pre> graph TD S((S)) --- 3+10.1 A((A)) S --- 4+9.2 D((D)) </pre>
A*, step 2	 <pre> graph TD S((S)) --- 13.1 A((A)) S --- 13.2 D1((D)) A --- 3+4+5.8 B((B)) A --- 3+5+9.2 D2((D)) </pre>
A*, step 3	 <pre> graph TD S((S)) --- 13.1 A((A)) S --- 13.2 D1((D)) A --- 12.8 B((B)) A --- 17.2 D2((D)) B --- 3+4+4+3.4 C((C)) B --- 3+4+5+7.1 E((E)) </pre>

<p>A*, step 4</p>	 <pre> graph TD S((S)) -- 13.1 --> A1((A)) S -- 13.2 --> D1((D)) A1 -- 12.8 --> B((B)) A1 -- 17.2 --> D2((D)) B -- 14.4 --> C((C)) B -- 19.1 --> E1((E)) D1 -- 4+2+7.1 --> E2((E)) D1 -- 4+5+10.1 --> A2((A)) </pre>
<p>A*, step 5</p>	 <pre> graph TD S((S)) -- 13.1 --> A1((A)) S -- 13.2 --> D1((D)) A1 -- 12.8 --> B((B)) A1 -- 17.2 --> D2((D)) B -- 14.4 --> C((C)) B -- 19.1 --> E1((E)) D1 -- 13.1 --> E2((E)) D1 -- 19.1 --> A2((A)) E2 -- 4+2+5+5.8 --> B2((B)) E2 -- 4+2+4+3.5 --> F((F)) </pre>
<p>A*, step 6</p>	 <pre> graph TD S((S)) -- 13.1 --> A1((A)) S -- 13.2 --> D1((D)) A1 -- 12.8 --> B((B)) A1 -- 17.2 --> D2((D)) B -- 14.4 --> C((C)) B -- 19.1 --> E1((E)) D1 -- 13.1 --> E2((E)) D1 -- 19.1 --> A2((A)) E2 -- 16.8 --> B2((B)) E2 -- 13.5 --> F((F)) F --> G((G)) </pre>

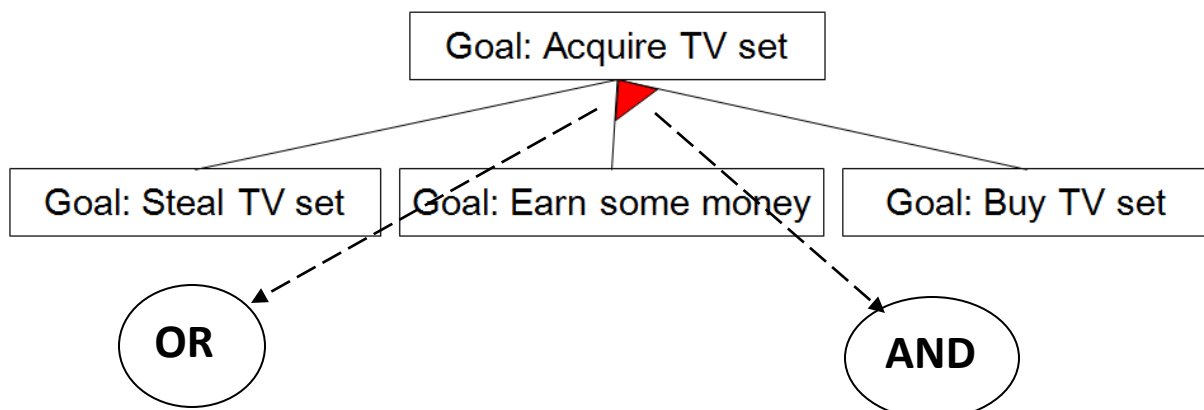
4. Problem Reduction:

In problem reduction, a complex problem is broken down or decomposed into a set of primitive sub problem; solutions for these primitive sub-problems are easily obtained. The solutions for all the sub problems collectively give the solution for the complex problem. The problem Reduction technique can be represented using **AndOR** Graph.



AndOR Graph :

To represent problem reduction techniques we need to use an AND-OR graph/tree. AND NODES successors must all be achieved. OR NODES where one of the successors must be achieved (i.e., they are alternatives). This decomposition or reduction, generates arcs that we call AND arcs. One AND arc may point to a number of successor nodes, all of which must be solved in order for the arc to point to a solution. Following is example of reduction of problem of Acquiring TV Set using ANDOR Graphs.



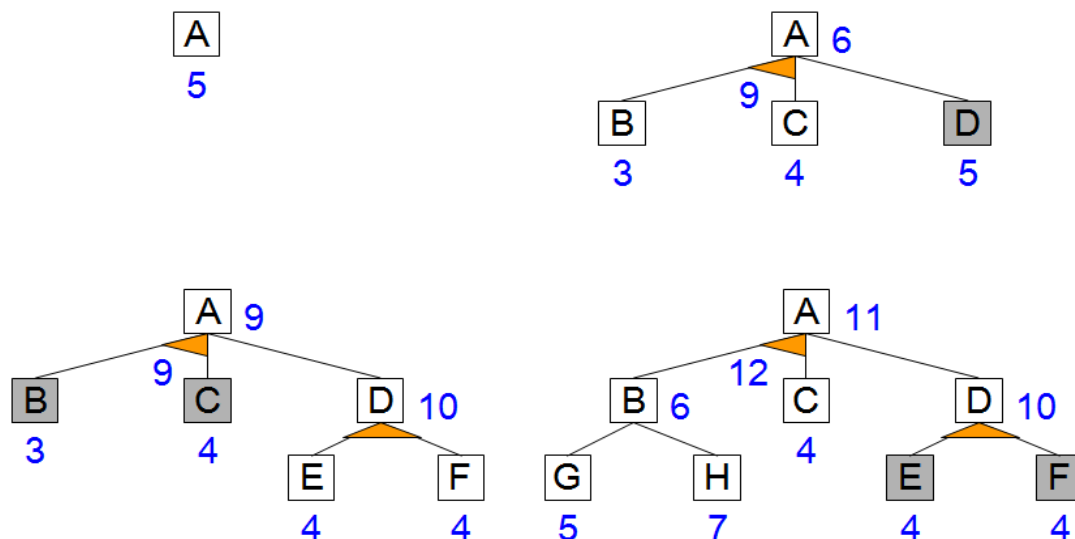
4.1 : AO* Algorithm: The AO* algorithm is problem reduction technique which makes use of AND and OR graph in order to obtain the solution. The AO* algorithm or Problem Reduction algorithm is given below :

Problem Reduction Algorithm using AndOR Graphs /AO* Algorithm

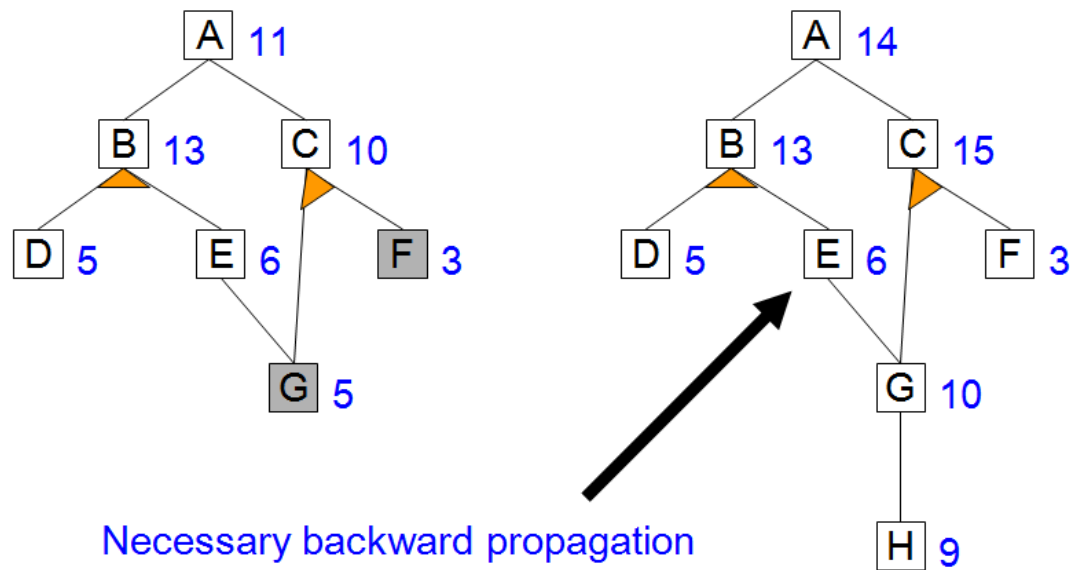
1. Start with initial node say N
2. Expand the node with all possible successors say $Ns_1, Ns_2, Ns_3, \dots, Ns_n$ each representing the *simple sub solutions*.
3. Categorize the Successors into AND Node successors and OR Node Successors.
4. Estimate the Objective function $f(N)$ w.r.t all solvable nodes of OR Node and AND Node successors.
 1. For AND Node $f(N) = \sum [f(Ns_i) + c(N, Ns_i)]$
 2. For OR Node $f(N) = \min \{f(Ns_i) + c(N, Ns_i)\}$ where Ns_i represents successor of N
5. Select the optimal successor which have solution .
6. Repeat the Steps from 2 for selected optimal successor until the goal is reached.

Example: Following are the two examples which illustrate the application of AO* algorithm for Problem Reduction .

Example 1



Example 2



Advantages and Disadvantages of AO* Algorithm

Advantages

- It is an optimal algorithm.
- If traverse according to the ordering of nodes.
- It can be used for both OR and AND graph.

Disadvantages

- Sometimes for unsolvable nodes, it can't find the optimal path.
- Its complexity is more than other algorithms.

Difference between A* and AO* Algorithm

- A* algorithm is a OR graph algorithm and
- AO* is a AND-OR graph algorithm.
- In OR graph algorithm it just find only one solution
- But in the AND-OR graph algorithm it finds more than one solution by ANDing two or more branches.

5. **Constraint Satisfaction Problem:** A CSP is a problem composed of a finite set of variables, each of which is associated with a *finite domain, and a set of constraints*. CSP is a search procedure that operates in a space of constraint state. Constraints in the initial state are originally given in the given problem. Any state is a goal state that has been constrained 'enough' where 'enough' means that each letter has been assigned a unique numeric value. The solution process in CSP contains cycles wherein the following things are done on each cycle:

- Propagate the constraints using rules that correspond to the arithmetic properties
- Guess a value for some letter whose value is not yet determined.

Constraint Satisfaction Algorithm

1. Propagate available constraints
 - a. All objects should be opened and must be assigned values in a complete solution
 - b. Repeat the succeeding steps until inconsistency or all objects assigned valid values
 - Select an object and strengthen this object as much as possible by a set of constraints that is applied to the object.
 - If the set of constraints is not matched from the previous set, then open all objects that share any of these constraints.
 - Remove selected objects
2. Return Solution, if the union of constraints exposed above defines a solution.
3. Otherwise return failure, if the union of constraints discovered above defines a contradiction

Example 1:

Cryptarithmic Problems: The problem is to plot integers from 0 to 9 for the alphabet specified in the crypt arithmetic problems with following constraints:

1. No two alphabets should have the same integer value.
2. Having allotted the different values for different alphabets we have to perform the arithmetic operations specified in the problem.

Example : *Assign numbers to letters so that the sum is correct as illustrated below :*

$$\begin{array}{r}
 \text{F O R T Y} \\
 + \quad \text{T E N} \\
 + \quad \text{T E N} \\
 \hline
 \text{S I X T Y}
 \end{array}$$

The steps used for solution is described below :

Step 1: Here $Y+N+N = Y$;

So N is either 0 or 5 Also , $T + E + E = T$;

So E is either 0 or 5 , Also E is 5 and N is 0.

So $c_1 = 0$, $c_2 = 1$

Step2 : $1 + R + T + T = X$ with a carry c_3 and $c_3 = 2$

$O + 2 = I$

Here , O must be a higher number

$O = 9$ and $I = 1$

$I + F = S$

T and R must be higher numbers

Then only we get a carry $c_3 = 2$

So $T = 8$ and $R = 7$

Step3: Just Assume $F = 2$, Then $S = 3$

$$\begin{array}{r}
 29786 \\
 + \quad 850 \\
 + \quad 850 \\
 \hline
 31486
 \end{array}$$

Solution

$F=2$, $O=9$

$R=7$, $T=8$

$Y=6$, $E=5$

$N=0$, $I=1$

$X=4$

Other Examples

SEND + MORE MONEY	8542 + 0915 09457	9567 + 1085 10652
DONALD + GERALD ROBERT	526485 + 197485 723970	
CROSS + ROADS DANGER	96233 + 62513 158746	

SOLUTION for the Problem SEND+MORE=MONEY

1) SEND + MORE = MONEY

$$\begin{array}{r}
 \begin{array}{ccccccc}
 & 5 & 4 & 3 & 2 & 1 & \\
 & S & E & N & D & & \\
 + & M & O & R & E & & \\
 & & c_3 & c_2 & c_1 & & \\
 \hline
 M & O & N & E & Y & &
 \end{array}
 \end{array}$$

- From Column 5, **M=1**, since it is only carry-over possible from sum of 2 single digit number in column 4.
- To produce a carry from column 4 to column 5 'S + M' is atleast 9 so '**S=8 or 9**' so '**S + M=9 or 10**' & so '**O = 0 or 1**'. But 'M=1', so '**O = 0**'.
- If there is carry from Column 3 to 4 then 'E=9' & so 'N=0'. But 'O = 0' so there is no carry & '**S=9**' & '**c3=0**'.
- If there is no carry from column 2 to 3 then 'E=N' which is Impossible, therefore there is carry & '**N=E+1**' & '**c2=1**'.
- If there is carry from column 1 to 2 then '**N+R=E mod 10**' & 'N=E+1' so 'E+1+R=E mod 10', so 'R=9' but 'S=9', so there must be carry from column 1 to 2. Therefore '**c1=1**' & '**R=8**'.
- To produce carry 'c1=1' from column 1 to 2, we must have 'D+E=10+Y' as Y cannot be 0/1 so D+E is atleast 12. As D is atmost 7 & E is at least 5 (D cannot be 8 or 9 as it is already assigned). N is atmost 7 & 'N=E+1' so 'E= 5 or 6'.

7. If E were 6 & D+E at least 12 then D would be 7, but 'N=E+1' & N would also be 7 which is impossible. Therefore 'E=5' & 'N=6'.

8. D+E is atleast 12 for that we get 'D=7' & 'Y=2'.

SOLUTION:

```
  9 5 6 7
+ 1 0 8 5
-----
1 0 6 5 2
```

VALUES:

S = 9 , E = 5 , N = 6 , D = 7 , M = 1 , O = 0 , R = 8 , Y = 2

Example2: Map-Coloring



- **Variables** *WA, NT, Q, NSW, V, SA, T*
- **Domains** $D_i = \{\text{red, green, blue}\}$
- **Constraints:** adjacent regions must have different colors
- **Solutions:** WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue and T = green

6. Means End Analysis: MEA is a technique that was first used in general problems solver by Newell and Simon. It is a problem-solving method in which the *solution with minimum difference between the current state and the goal state is determined*. The MEA algorithm is given below:

Means-Ends Analysis Algorithm

1. Until the goal is reached or no more procedures are available,
 - a) Describe the current state, the goal state, and Calculate the difference between the two.
 - b) Use the difference, to select a promising procedure.
2. Use the promising procedure and update the current state.
3. If the goal is reached, announce success; otherwise, announce failure.

Example: Consider the Robot Task in which a robot moves a desk with two things on it from one room to another. The different operator that the Robot can manipulate are PUSH, CARRY, WALK, PICKUP, PUTDOWN and PLACE. The preconditions and results of these operators are described in figure below.

A Robot's Operators		
<i>Operator</i>	<i>Preconditions</i>	<i>Results</i>
PUSH(obj, loc)	at(robot,obj)^ large(obj)^ clear(obj)^ armempty	at(obj, loc)^ at(robot, loc)
CARRY(obj, loc)	at(robot,obj) ^ small(obj)	at(obj, loc)^ at(robot, loc)
WALK(loc) PICKUP(obj)	none at(robot, obj)	at(robot, loc) holding(obj)
PUTDOWN(obj)	holding(obj)	\neg holding(obj)
PLACE(obj1, obj2)	at(robot, obj2)^ holding(obj1)	on(obj1, obj2)

The preconditions for the PUSH operator is Robot must be in front of the object, the object must be large, object must be clear that is no other object must be placed on the given object and the arm of the robot must be empty or free. The preconditions for CARRY is that robot and object must be present and the object

must be small. The precondition for PICKUP is the robot must be in front of the object and so on. The objective the means end analysis is to reduce the difference between the goal state and present state. The difference table used for the above robot problem is given below:

Operators

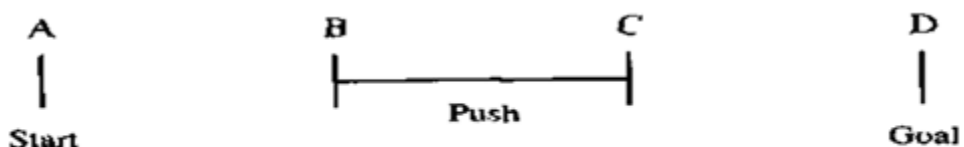
	Push	Carry	Walk	Pickup	Putdown	Place
Move object	*	*				
Move robot			*			
Clear object				*		
Get object on object						*
Get arm empty					*	*
Be holding object				*		

↑
differences

In order to move a object the robot can select Push or Carry operator. IF the robot wants to move from one place to another place it can select Move Robot. In order clear the object the operator selected is Pickup. In order to get object on the other object Place is used. In order to empty the arm one has to use place operator and putdown operator.

Suppose that the robot in this domain were given the problem of moving a desk with two things on it from one room to another. The objects on top must be moved. The main difference between the start state and the goal state would be the location of the desk. To reduce this difference either PUSH or carry could be chosen. If CARRY is chosen first, its precondition must be met. This results in two more differences that must be reduced: the location of the robot and the size of the desk. The location of the robot can be handled by applying WALK, but there re no operators than can change the size of an object. So, this path leads to a dead end.

Following the other branch, we attempt to apply PUSH. Figure below shows the problem solver's progress at this point.



The Progress of the Means-Ends Analysis Method

It has found a way of doing something useful. But it does not lead to the goal state. The proper sequence of the operator to be selected is *WALK* , *PICKUP* , *PUTDOWN* ,

PICKUP , *PUTDOWN*, *PUSH*, *WALK* , *PICKUP*, *PLACE*, *WALK* , *PICKUP* , *PLACE* . The progress of the Means Ends Method. After each step, the difference towards the goal decreases.

A					B		C			D		
WALK	PICKUP	PUTDOWN	PICKUP	PUTDOWN	PUSH	WALK	PICKUP	PLACE	WALK	PICKUP	PLACE	

START GOAL

The process can be summarized and put in the form of algorithm as given below :

Algorithm: Means-Ends Analysis (CURRENT, GOAL)

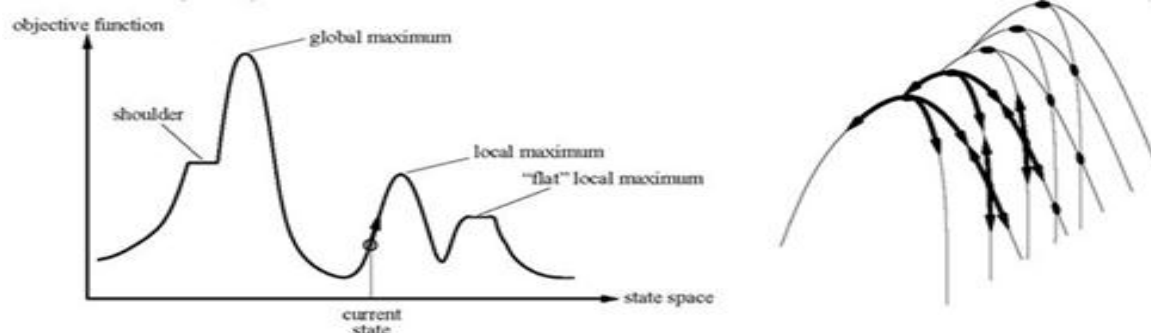
1. Compare CURRENT with GOAL. If there are no differences between them then return.
2. Otherwise, select the most important difference and reduce it doing the following until success or failure is signaled:
 - a. Select an as yet untried operator O that is applicable to the current difference. If there are no such operators, then signal failure.
 - b. Attempt to apply O to CURRENT. Generate descriptions of two states: O-START, a state in which O's preconditions are satisfied and ORESULT, the state that would result if O were applied in O-START.
 - c. If (FIRST-PART <- MEA (CURRENT, O-START)) and (LAST-PART <- MEA (O-RESULT, GOAL)) are successful, then signal success and return the result of concatenating FIRST-PART,O, and LAST-PART.

Q8. What are the Problems With hill climbing? What are the possible solutions?

Answer: The problems with hill climbing techniques are :

1. **Local maximum, or the foothill problem:** there is a peak, but it is lower than the highest peak in the whole space.
2. **The plateau problem:** all local moves are equally unpromising, and all peaks seem far away.
3. **The ridge problem:** almost every move takes us down.

The above problems are illustrated in the figure below:



Solutions to Problem

- *Backtrack to some earlier node* and try going in a different direction
- *Make a big jump in some direction* to try to get to a new section of the search space.
- *Apply two or more rules before doing the test.* This corresponds to moving in several directions at once.