Q1. With suitable examples, explain the various problem characteristics of AI problem.

Ans:

- Is the problem decomposable into a set of (nearly) independent smaller or easier subproblems?
- Can solution steps be ignored or at least undone if they prove unwise?
- Is the problem's universe predictable?
- Is a good solution to the problem obvious without comparison to all other possible solutions?
- Is the desired solution a state of the world or a path to a state?
- Is a large amount of knowledge absolutely required to solve the problem, or is knowledge important only to constrain the search?
- Can a computer that is simply given the problem return the solution, or will the solution of the problem require interaction between the computer and a person?

1) IS YOUR PROBLEM DECOMPOSABLE?

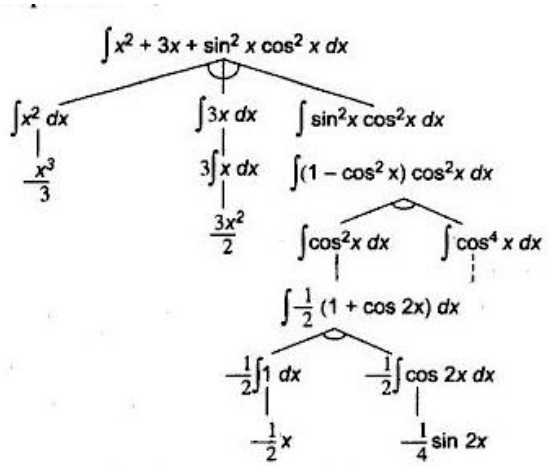


Fig. 2.9 *A Decomposable Problem*

$$\int (x^2 + 3x + \sin^{2*} \cdot \cos^2 x)\, dx$$

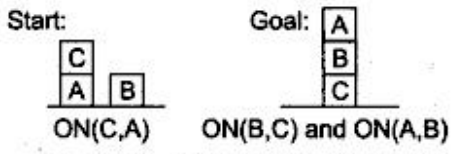


Fig. 2.10 *A Simple Blocks World Problem*

2. Can Solution Steps Be Ignored Or Undone

The 8-Puzzle: The 8-puzzle is a square tray in which are placed, eight square tiles. The remaining ninth square is uncovered. Each tile has a number on it. A tile that is adjacent to the blank space can be slid into that space. A game consists of a starting position and a specified goal position. The goal is to transform the starting position into the goal position by sliding the tiles around.

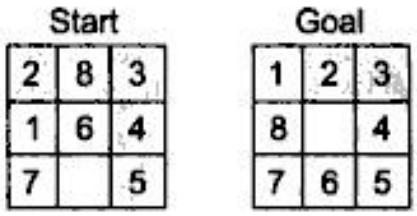


Fig. 2.12 *An Example of the 8-Puzzle*

These three problems—theorem proving, the 8-puzzle, and chess—illustrate the differences between three important classes of problems:

- Ignorable (e.g., theorem proving), in which solution steps can be ignored
- Recoverable (e.g., 8-puzzle), in which solution steps can be undone
- Irrecoverable (e.g., chess), in which solution steps cannot be undone

3)IS THE UNIVERSE PREDICTABLE?

Again suppose that we are playing with the 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. We can use planning to avoid having to undo actual moves, although it will still be necessary to backtrack past those moves one at a time during the planning process. Thus a control structure that allows backtracking will be necessary.

However, in games other than the 8-puzzle, this planning process may not be possible. Suppose we want to play bridge. One of the decisions we will have to make is which card to play on the first trick. What we would like to do is to plan the entire hand before making that first play. But now it is not possible to do such planning with certainty since We cannot know exactly where all the cards are or what the other players will do on their turns. The best we can do is to investigate several plans and use probabilities of the various outcomes to choose a plan that has the highest estimated probability of leading to a good score on the hand.

- Playing bridge. But we can do fairly well since we have available accurate estimates of the probabilities of each of the possible outcomes.
- Controlling a robot arm. The outcome is uncertain for a variety of reasons. Someone might move something into the path of the arm. The gears of the arm might stick. A slight error could cause the arm to knock over a whole stack of things.
- Helping a lawyer decide how to defend his client against a murder charge. Here we probably cannot even list all the possible outcomes, much less assess their probabilities.

4) IS THE GOOD SOLUTION ABSOLUTE OR RELATIVE?

Consider the problem of answering questions 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 Pompeians 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 the question "Is Marcus alive?"

	Justification
1. Marcus was a man.	axiom 1
4. All men are mortal.	axiom 4
8. Marcus is mortal.	1, 4
3. Marcus was born in 40 A.D.	axiom 3
7. It is now 1991 A.D.	axiom 7
9. Marcus' age is 1951 years.	3, 7
6. No mortal lives longer than 150 years.	axiom 6
10. Marcus is dead.	8, 6, 9
OR	
7. It is now 1991 A.D.	axiom 7
5. All Pompeians died in 79 A.D.	axiom 5
11. All Pompeians are dead now.	7, 5
2. Marcus was a Pompeian.	axiom 2
12. Marcus is dead.	11, 2

Fig. 2.13 *Two Ways of Deciding That Marcus Is Dead*

But now consider again the traveling salesman problem. Our goal is to find the shortest route that visits each city exactly once. Suppose the cities to be visited and the distances between them are as shown in Fig. 2.14.

	Boston	New York	Miami	Dallas	S.F.
Boston		250	1450	1700	3000
New York	250		1200	1500	2900
Miami	1450	1200		1600	3300
Dallas	1700	1500	1600		1700
S.F.	3000	2900	3300	1700	

Fig. 2.14 *An Instance of the Traveling Salesman Problem*

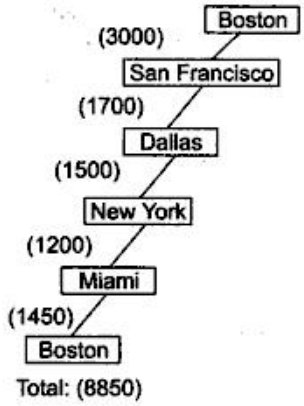


Fig. 2.15 *One Path among the Cities*

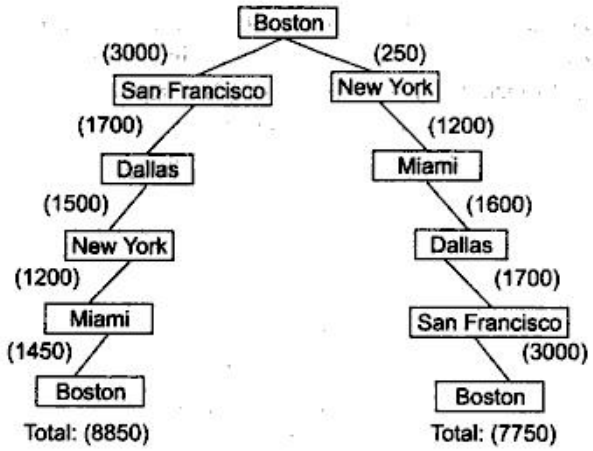


Fig. 2.16 *Two Paths Among the Cities*

5)IS THE SOLUTION A STATE OR A PATH?

Consider the problem of finding a consistent interpretation for the sentence

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

The word "bank" may refer either to a financial institution or to a side of a river. But only one of these may have a president.

- The word "dish" is the object of the verb "eat." It is possible that a dish was eaten. But it is more likely that the pasta salad in the dish was eaten.
- Pasta salad is a salad containing pasta. But there are other ways meanings can be formed from pairs of nouns. For example, dog food does not normally contain dogs.
- The phrase "with the fork" could modify several parts of the sentence. In this case, it modifies the verb "eat." But, if the phrase had been "with vegetables," then the modification structure would be different. And if the phrase had been "with her friends," the structure would be different still.

6) WHAT IS THE ROLE OF KNOWLEDGE?

Consider again the problem of playing chess. Suppose you had unlimited computing power available. How much knowledge would be required by a perfect program? The answer to this question is very little—just the rules for determining legal moves and some simple control mechanism that implements an appropriate search procedure. Additional knowledge about such things as good strategy and tactics could of course help considerably to constrain the search and speed up the execution of the program.

7) DOES THE TASK REQUIRE INTERACTION WITH A PERSON?

Sometimes it is useful to program computers to solve problems in ways that the majority of people would not be able to understand. This is fine if the level of the interaction between the computer and its human users is problem-in solution-out. But increasingly we are building programs that require intermediate interaction with people, both to provide additional input to the program and to provide additional reassurance to the user.

- Solitary, in which the computer is given a problem description and produces an answer with no intermediate communication and with no demand for an explanation of the reasoning process
- Conversational, in which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or to provide additional information to the user, or both

Q2. Define AI? Outline nature of problem that led to the development of AI? Explain typical task domains of AI?

Ans:- The Foundations of AI:- Philosophy (423 BC - present):

(a) Logic, methods of reasoning.

(b)Mind as a physical system.

(c) Foundations of learning, language, and rationality.

- Mathematics (c.800 - present):
  - Formal representation and proof.
  - Algorithms, computation, decidability, tractability.
  - Probability.
- Psychology (1879 - present):
  - Adaptation.
  - Phenomena of perception and motor control.
  - Experimental techniques.
- Linguistics (1957 - present):
  - Knowledge representation.
  - Grammar.

Task Domains of AI:-

Mundane Tasks:

- Perception
  - Vision
  - Speech
- Natural Languages
  - Understanding
  - Generation
  - Translation
- Common sense reasoning
- Robot Control

- Formal Tasks
  - Games : chess, checkers etc
  - Mathematics: Geometry, logic,Proving properties of programs
- Expert Tasks:
  - Engineering (Design, Fault finding, Manufacturing planning)
  - Scientific Analysis
  - Medical Diagnosis
  - Financial Analysis

Q3. Define AI technique and give few of them.

Ans: AI Technique:- Intelligence requires Knowledge, knowledge posesses less desirable properties such as:

- Voluminous
- Hard to characterize accurately
- Constantly changing
- Differs from data that can be used

AI technique is a method that exploits knowledge that should be represented in such a way that:

- Knowledge captures generalization
- It can be understood by people who must provide it
- It can be easily modified to correct errors.
- It can be used in variety of situations

Introductory Problem: Tic-Tac-Toe

Program 1:

Data Structures:

- Board: 9 element vector representing the board, with 1-9 for each square. An element contains the value 0 if it is blank, 1 if it is filled by X, or 2 if it is filled with a O
- Movetable: A large vector of 19,683 elements (3^9), each element is 9-element vector.

Algorithm:

1. View the vector as a ternary number. Convert it to a decimal number.
2. Use the computed number as an index into Move-Table and access the vector stored there.
3. Set the new board to that vector.

Comments:

This program is very efficient in time.

1. A lot of space to store the Move-Table.
2. A lot of work to specify all the entries in the Move-Table.
3. Difficult to extend.

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

Program 2:

Data Structure: A nine element vector representing the board. But instead of using 0,1 and 2 in each element, we store 2 for blank, 3 for X and 5 for O

Functions:

Make2: returns 5 if the center sqaure is blank. Else any other balnk sq

Posswin(p): Returns 0 if the player p cannot win on his next move; otherwise it returns the number of the square that constitutes a winning move. If the product is 18 (3x3x2), then X can win. If the product is 50 (5x5x2) then O can win.

Go(n): Makes a move in the square n

Strategy:

Turn = 1 Go(1)

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) $\neq$ 0, then Go(Posswin(X))

Comments:

1. Not efficient in time, as it has to check several conditions before making each move.
2. Easier to understand the program's strategy.
3. Hard to generalize.

8	**3**	**4**
1	**5**	**9**
6	**7**	**2**

15 - (8 + 5)

Comments:

1. Checking for a possible win is quicker.
2. Human finds the row-scan approach easier, while computer finds the number-counting approach more efficient.

Program 3:

1. If it is a win, give it the highest rating.
2. Otherwise, consider all the moves the opponent could make next. Assume the opponent will make the move that is worst for us. Assign the rating of that move to the current node.
3. The best node is then the one with the highest rating.

Comments:

1. Require much more time to consider all possible moves.
2. Could be extended to handle more complicated games.

Introductory Problem: Question Answering

"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 favourite dress".

Q1: What did Mary go shopping for?

Q2: What did Mary find that she liked?

Q3: Did Mary buy anything?

Program 1:

1. Match predefined templates to questions to generate text patterns.

2.Match text patterns to input texts to get answers.

"What did X Y" "What did Mary go shopping for?"

"Mary go shopping for Z"

Z = a new coat

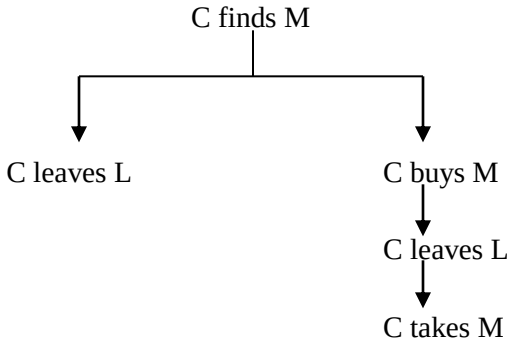
Program 2:

Structured representation of sentences:

Event2:		Thing1:	
instance:	Finding	instance:	Coat
tense:	Past	colour:	Red
agent:	Mary		
object:	Thing 1		

Program 3:

Background world knowledge:



Q4. Explain different categories of Production system.

Ans:- A Knowledge representation formalism consists of collections of condition-action rules(Production Rules or Operators), a database which is modified in accordance with the rules, and a Production System Interpreter which controls the operation of the rules i.e The 'control mechanism' of a Production System, determining the order in which Production Rules are fired.

A system that uses this form of knowledge representation is called a production system.

Example:-

IF the initial state is a goal state THEN quit.

The major components of an AI production system are

i. A global database

ii. A set of production rules and

iii. A control system

The goal database is the central data structure used by an AI production system. The production system. The production rules operate on the global database. Each rule has a precondition that is either satisfied or not by the database. If the precondition is satisfied, the rule can be applied. Application of the rule changes the database. 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.

Four classes of production systems:-

1. A monotonic production system

2. A non monotonic production system

3. A partially commutative production system

4. A commutative production system.

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 the a recording of an expert thinking out loud.

Disadvantages of Production Systems:-

One important disadvantage is the fact that it may be very difficult analyse the flow of control within a production system because the individual rules don't call each other.

Production systems describe the operations that can be performed in a search for a solution to the problem. They can be classified as follows.

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.

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.

Theorem proving falls under monotonic partially communicative system. Blocks world and 8 puzzle problems like chemical analysis and synthesis come under monotonic, not partially commutative systems. Playing the game of bridge comes under non monotonic , not partially commutative system.

For any problem, several production systems exist. Some will be efficient than others. Though it may seem that there is no relationship between kinds of problems and kinds of production systems, in practice there is a definite relationship.
Partially commutative , monotonic production systems are useful for solving ignorable problems. These systems are important for man implementation standpoint because they can be implemented without the ability to backtrack to previous states, when it is discovered that an incorrect path was followed. Such systems increase the efficiency since it is not necessary to keep track of the changes made in the search process.
Monotonic partially commutative systems are useful for problems in which changes occur but can be reversed and in which the order of operation is not critical (ex: 8 puzzle problem).
Production systems that are not partially commutative are useful for many problems in which irreversible changes occur, such as chemical analysis. When dealing with such systems, the order in which operations are performed is very important and hence correct decisions have to be made at the first time itself.

Q5 Explain Production system characteristics.
Ans:-

- A set of rules, each consisting of a left side (a pattern) that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.[3]
- One or more knowledge/databases that contain whatever information is appropri- ate for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem. The information in these databases may be structured in any appropriate way.
- A control strategy that 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.
- A rule applier.

Q6. What are the steps involved in designing to solve AI Problem?
Ans: **Steps involved in designing a program to solve an AI problem:**
To build a system to solve particular problem we need four steps:-

(i) Define the problem precisely:- This definition must include precise specification of what the initial situations will be as well as what final situations constitute acceptable to solution of problem.
(ii) Analyze the problem:- A few important features can have a immense impact on the appropriateness of various possible technique for solving the problem.
(iii) Isolate and represent the task knowledge that is necessary to solve the problem.
(iv) Choose the best problem solving technique and apply it to the particular problem.

Q7. What is difference between simple hill climbing, steepest hill climbing and simulated annealing.
Ans:-

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. In a pure generate-and-test procedure, the test function responds with only a yes or no. But if the test function is augmented with a heuristic function[2] that provides an estimate of how close a given state is to a goal state, the generate procedure can exploit it as shown in the procedure below. This is particularly nice because often the computation of the heuristic function can be done at almost no cost at the same time that the test for a solution is being performed. Hill climbing is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available. For example, suppose you are in an unfamiliar city without a map and you want to get downtown. You simply aim for the tall buildings. The heuristic function is just distance between the current location and the location of the tall buildings and the desirable states are those in which this distance is minimized.

- SIMPLE HILL CLIMBING
- STEEPEST ASCENT HILL CLIMBING
- SIMULATED ANNEALING

Simple Hill Climbing:-

Algorithm: Simple 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 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.

"Is one state *better* than another?"

STEEPEST ASCENT HILL CLIMBING:-

A useful variation on simple hill climbing considers all the moves from the current state and selects the best one as the next state. This method is called *steepest-ascent hill climbing* or *gradient search.* Notice that this contrasts with the basic method in which the first state that is better than the current state is selected. The algorithm works as follows.

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 *SUCC* be a state such that any possible successor of the current state will be better than *SUCC.*
   (b) For each operator that applies to the current state do:
      (i) Apply the operator and generate a new state.
      (ii) Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to *SUCC.* If it is better, then set *SUCC* to this state. If it is not better, leave *SUCC* alone.
   (c) If the *SUCC* is better than current state, then set current state to *SUCC.*

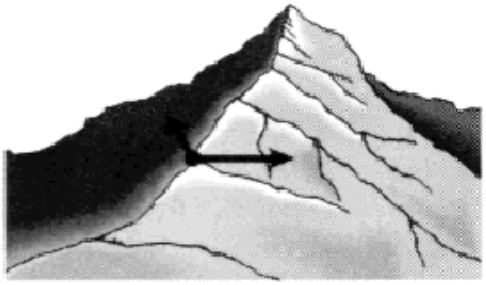
DISADVANTAGES OF STEEPEST ASCENT HILL CLIMBING:-

A *local maximum* is a state that is better than all its neighbors but is not better than some other states farther away. At a local maximum, all moves appear to make things worse. Local maxima are particularly frustrating because they often occur almost within sight of a solution. In this case, they are called *foothills.*

A plateau is a flat area of the search space in which a whole set of neighboring states have the same value. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.

A *ridge* is a special kind of local maximum. It is an area of the search space that is higher than surrounding areas and that itself has a slope (which one would like to climb). But the orientation of the high region, compared to the set of available moves and the directions in which they move, makes it impossible to traverse a ridge by single moves.

(A) **(B)**



(C)

Figure:- (A)Local Maxima (B) Plateau (C) Ridges

Probable Solution:-

There are some ways of dealing with these problems, although these methods are by no means guaranteed:

- Backtrack to some earlier node and try going in a different direction. This is particularly reasonable if at that node there was another direction that looked as promising or almost as promising as the one that was chosen earlier. To implement this strategy, maintain a list of paths almost taken and go back to one of them if the path that was taken leads to a dead end. This is a fairly good way of dealing with local maxima.
- Make a big jump in some direction to try to get to a new section of the search space. This is a particularly good way of dealing with plateaus. If the only rules available describe single small steps, apply them several times in the same direction.
- Apply two or more rules before doing the test. This corresponds to moving in several directions at once. This is a particularly good strategy for dealing with ridges.

SIMULATED ANNEALING:-

Simulated annealing [Kirkpatrick *et al.*, 1983] as a computational process is patterned after the physical process *of annealing*, in which physical substances such as metals are melted (i.e., raised to high energy levels) and then gradually cooled until some solid state is reached. The goal of this process is to produce a minimal-energy final state. Thus this process is one of valley descending in which the objective function is the energy level. Physical substances usually move from higher energy configurations to lower ones, so the valley descending occurs naturally. But there is some probability that a transition to a higher energy state will occur. This probability is given by the function

$$p = e^{-\Delta E/kT}$$

The rate at which the system is cooled is called the *annealing schedule*. Physical annealing processes are very sensitive to the annealing schedule. If cooling occurs too rapidly, stable regions of high energy will form.

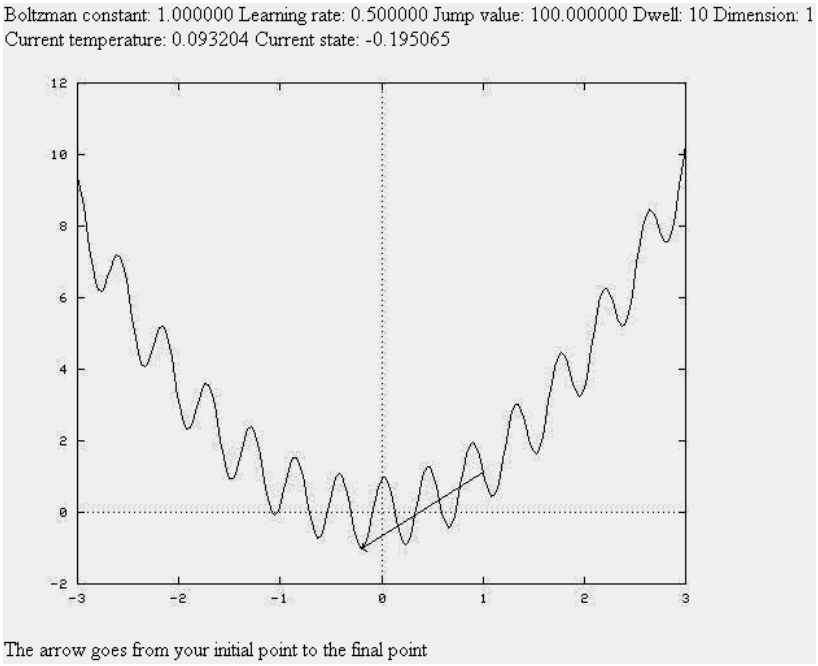


Figure:- Simulated Annealing

Differences:-

- The algorithm for simulated annealing is 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 good idea to maintain, in addition to the current state, the best state found so far.

Q8. What are the steps involved in designing a program to solve an AI problem?

Ans:- To build a system to solve particular problem we need four steps:-

(iv) Define the problem precisely:- This definition must include precise specification of what the initial situations will be as well as what final situations constitute acceptable to solution of problem.

(v) Analyze the problem:- A few important features can have a immense impact on the appropriateness of various possible technique for solving the problem.

(vi) Isolate and represent the task knowledge that is necessary to solve the problem.

(vii) Choose the best problem solving technique and apply it to the particular problem.

Q9. What is state space? Give state space representation for water-jug problem .

Ans:- Defining state space of the problem: A set of all possible states for a given problem is known as state space of the problem. Representation of states is highly beneficial in AI because they provide all possible states, operations and the goals. If the entire sets of possible states are given, it is possible to trace the path from the initial state to the goal state and identify the sequence of operators necessary for doing it.

A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jug?

The state space is described by the pair (m; n), the volumes in the 4pt and 3pt can respectively.

The initial state is (0; 0), and the final state is (2; 0). Here then we already see an example of multiple goal states.

A set of production rules might be:

1	(x, y) if $x < 4$	$\rightarrow$ $(4, y)$	Fill the 4-gallon jug
2	(x, y) if $y < 3$	$\rightarrow$ $(x, 3)$	Fill the 3-gallon jug
3	(x, y) if $x > 0$	$\rightarrow$ $(x - d, y)$	Pour some water out of the 4-gallon jug
4	(x, y) if $y > 0$	$\rightarrow$ $(x, y - d)$	Pour some water out of the 3-gallon jug
5	(x, y) if $x > 0$	$\rightarrow$ $(0, y)$	Empty the 4-gallon jug on the ground
6	(x, y) if $y > 0$	$\rightarrow$ $(x, 0)$	Empty the 3-gallon jug on the ground
7	(x, y) if $x + y \geq 4$ and $y > 0$	$\rightarrow$ $(4, y - (4 - x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full
8	(x, y) if $x + y \geq 3$ 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) if $x + y \leq 4$ and $y > 0$	$\rightarrow$ $(x + y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10	(x, y) if $x + y \leq 3$ and $x > 0$	$\rightarrow$ $(0, x + y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11	$(0, 2)$	$\rightarrow$ $(2, 0)$	Pour the 2 gallons from the 3-gallon jug into the 4-gallon jug
12	$(2, y)$	$\rightarrow$ $(0, y)$	Empty the 2 gallons in the 4-gallon jug on the ground

Fig. 2.3 *Production Rules for the Water Jug Problem*

Solutions:-

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

Fig. 2.4 *One Solution to the Water Jug Problem*

FORMAL DESCRIPTION OF A PROBLEM:-

1. Define a state space that contains all the possible configurations of the relevant objects (and perhaps some impossible ones). It is, of course, possible to define this space without explicitly enumerating all of the states it contains.
2. Specify one or more states within that space that describe possible situations from which the problem-solving process may start. These states are called the *initial states.*

3. Specify one or more states that would be acceptable as solutions to the problem. These states are called *goal states.*
4. Specify a set of rules that describe the actions (operators) available. Doing this will require giving thought to the following issues:
   - What unstated assumptions are present in the informal problem description?
   - How general should the rules be?
   - How much of the work required to solve the problem should be precomputed and represented in the rules?

Q10. Explain Production System in brief.

Ans: Production Systems

A production system is a system that adapts a system with production rules.

A *production system* consists of:

• A set of rules, each consisting of a left side and a right hand side. Left hand side or pattern determines the applicability of the rule and a right side describes the operation to be performed if the rule is applied.

• One or more knowledge/databases that contain whatever information is appropriate for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem. The information in these databases may be structured in any appropriate way.

• A control strategy that 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.

• A rule applier.

Q11. Explain BFS and DFS algorithm in brief.

Ans:- **BREADTH- FIRST SEARCH:**

Algorithm: Breadth-First Search

1. Create a variable called *NODE-LIST* and set it to the initial state.
2. Until a goal state is found or *NODE-LIST* is empty:
   (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 the new state is a goal state, quit and return this state.
      (iii) Otherwise, add the new state to the end of *NODE-LIST.*

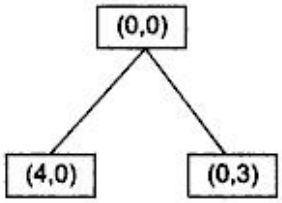


Fig. 2.5 *One Level of a Breadth-First Search Tree*

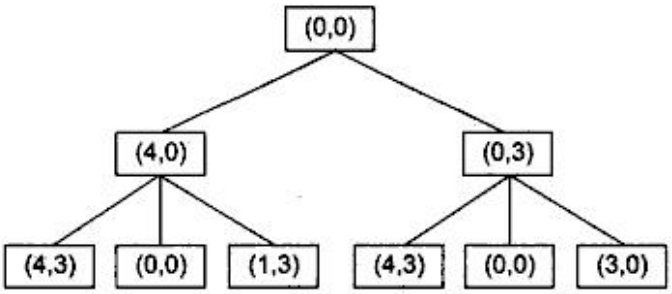


Fig. 2.6 *Two Levels of a Breadth-First Search Tree*

Advantages of Breadth-First Search

- Breadth-first search will not get trapped exploring a blind alley. This contrasts with depth-first searching, which may follow a single, unfruitful path for a very long time, perhaps forever, before the path actually terminates in a state that has no successors. This is a particular problem in depth-first search if there are loops (i.e., a state has a successor that is also one of its ancestors) unless special care is expended to test for such a situation. The example in Fig. 2.7, if it continues always choosing the first (in numerical sequence) rule that applies, will have exactly this problem.
- If there is a solution, then breadth-first search is guaranteed to find it. Furthermore, if there are multiple solutions, then a minimal solution (i.e., one that requires the minimum number of steps) will be found. This is guaranteed by the fact that longer paths are never explored until all shorter ones have already been examined. This contrasts with depth-first search, which 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.

DEPTH- FIRST SEARCH:

Algorithm: Depth-First Search

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 the initial state. If there are no more successors, signal failure.
   (b) Call Depth-First Search with *E* as the initial state.
   (c) If success is returned, signal success. Otherwise continue in this loop.

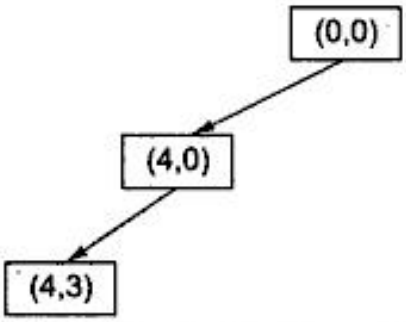


Fig. 2.7 *A Depth-First Search Tree*

Advantages of Depth-First Search

- Depth-first search requires less memory since only the nodes on the current path are stored. This contrasts with breadth-first search, where all of the tree that has so far been generated must be stored.
- By chance (or if care is taken in ordering the alternative successor states), depth-first search may find a solution without examining much of the search space at all. This contrasts with breadth-first search in which all parts of the tree must be examined to level n before any nodes on level $n + 1$ can be examined. This is particularly significant if many acceptable solutions exist. Depth-first search can stop when one of them is found.

Q12. Explain Best-First search method.

Ans:- (i)Depth-first search: not all competing branches having to be expanded.

(ii)Breadth-first search: not getting trapped on dead-end paths.

Combining the two is to follow a single path at a time, but switch paths whenever some competing path look more promising than the current one.

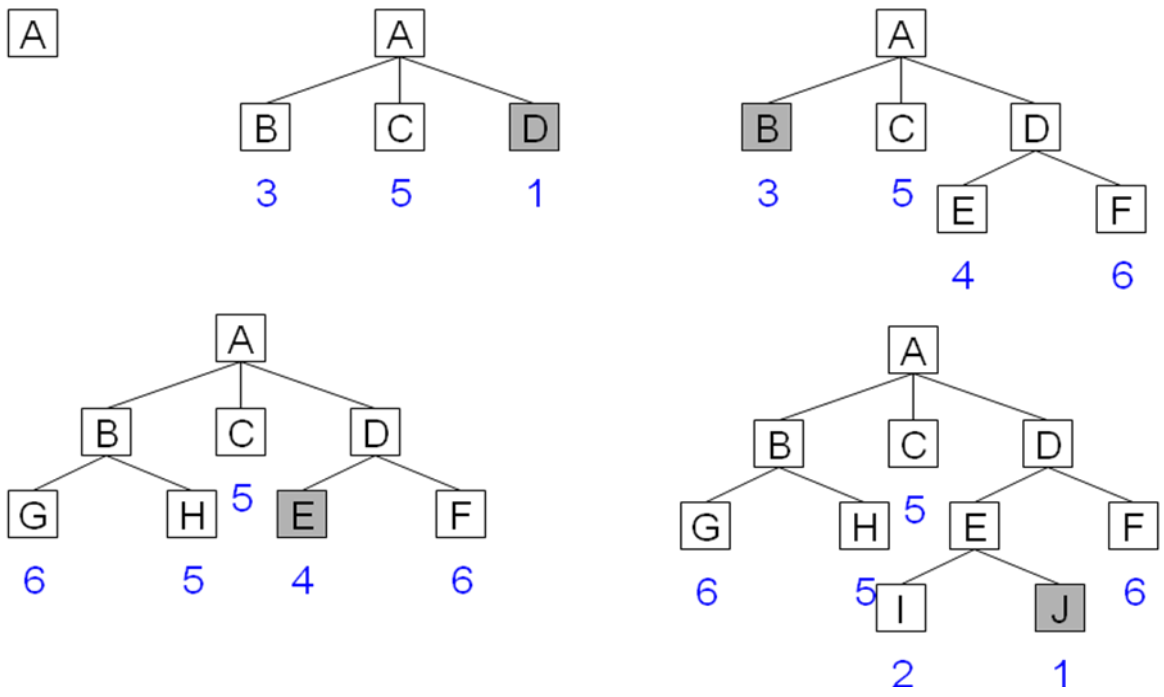


Figure:- Best First Search

- **OPEN:** nodes that have been generated, but have not examined. This is organized as a priority queue.
- **CLOSED:** nodes that have already been examined. Whenever a new node is generated, check whether it has been generated before.

Algorithm:-

1. OPEN = {initial state}.
2. Loop until a goal is found or there are no nodes left in OPEN:
   (a)Pick the best node in OPEN
   (b) Generate its successors
   (c) For each successor:
3. new → evaluate it, add it to OPEN, record its parent
   generated before → change parent, update successors

- Greedy search: h(n) = estimated cost of the cheapest path from node n to a goal state. It is neither optimal nor complete.

- Uniform-cost search: g(n) = cost of the cheapest path from the initial state to node n. Optimal and complete, but very inefficient.

Q13. Explain A* Algorithm with example.

Ans: A* algorithm:

The best-first search algorithm that was just presented is a simplification of an algorithm called A*, which was first presented by Hart *et al.* [1968; 1972]. This algorithm uses the same f', g, and h' functions, as well as the lists *OPEN* and *CLOSED*, that we have already described.

Algorithm: A*

1. Start with *OPEN* containing only the initial node. Set that node's g value to 0, its h' value to whatever it is, and its f' value to $h' + 0$, or h'. Set *CLOSED* to the empty list.
2. Until a goal node is found, repeat the following procedure: If there are no nodes on *OPEN*, report failure. Otherwise, pick the node on *OPEN* with the lowest f' value. Call it *BESTNODE*. Remove it from *OPEN*. Place it on *CLOSED*. See if *BESTNODE* is a goal node. If so, exit and report a solution (either *BESTNODE* if all we want is the node or the path that has been created between the initial state and *BESTNODE* if we are interested in the path). Otherwise, generate the successors of *BESTNODE* but do not set *BESTNODE* to point to them yet. (First we need to see if any of them have already been generated.) For each such *SUCCESSOR*, do the following:

(a) Set *SUCCESSOR* to point back to *BESTNODE*. These backwards links will make it possible to recover the path once a solution is found.

(b) Compute $g(SUCCESSOR) = g(BESTNODE)$ + the cost of getting from *BESTNODE* to *SUCCESSOR*.

(c) See if *SUCCESSOR* is the same as any node on *OPEN* (i.e., it has already been generated but not processed). If so, call that node *OLD*. Since this node already exists in the graph, we can throw *SUCCESSOR* away and add *OLD* to the list of *BESTNODE's* successors. Now we must decide whether *OLD's* parent link should be reset to point to *BESTNODE*. It should be if the path we have just found to *SUCCESSOR* is cheaper than the current best path to *OLD* (since *SUCCESSOR* and *OLD* are really the same node). So see whether it is cheaper to get to *OLD* via its current parent or to *SUCCESSOR* via *BESTNODE* by comparing their g values. If *OLD* is cheaper (or just as cheap), then we need do nothing. If *SUCCESSOR* is cheaper, then reset *OLD's* parent link to point to *BESTNODE*, record the new cheaper path in $g(OLD)$, and update f'(OLD).

(d) If *SUCCESSOR* was not on *OPEN*, see if it is on *CLOSED*. If so, call the node on *CLOSED OLD* and add *OLD* to the list of *BESTNODE's*, successors. Check to see if the new path or the old path is better just as in step *2(c)*, and set the parent link-and g and f' values appropriately. If we have just found a better path to *OLD*, we must propagate the improvement to *OLD's* successors. This is a bit tricky. *OLD* points to its successors. Each successor in turn points to its successors, and so forth, until each branch terminates with a node that either is still on *OPEN* or has no successors. So to propagate the new cost downward, do a depth-first traversal of the tree starting at *OLD*, changing each node's g value (and thus also its f' value), terminating each branch when you reach either a node with no successors or a node to which an equivalent or better path has already been found.[4] This condition is easy to check for. Each node's parent link points back to its best known parent. As we propagate down to a node, see if its parent points to the node we are coming from. If so, continue the propagation. If not, then its g value already reflects the better path of which it is part. So the propagation may stop here. But it is possible that with the new value of g being propagated downward, the path we are following may become better than the path through the current parent. So compare the two. If the path through the current parent is still better, stop the propagation. If the path we are propagating through is now better, reset the parent and continue propagation.

(e) If *SUCCESSOR* was not already on either *OPEN* or *CLOSED*, then put it on *OPEN*, and add it to the list of *BESTNODE's* successors. Compute $f'(.SUCCESSOR) = g(SUCCESSOR) + h'(SUCCESSOR)$.

Q14. Explain AO* Algorithm in brief with example.

Ans: AO* Algorithm:

AND-OR Graphs:-

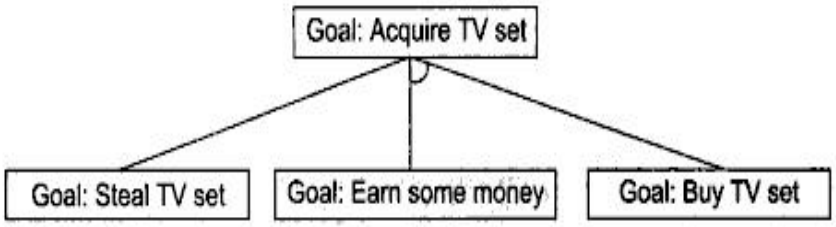


Fig. 3.6 *A Simple AND-OR Graph*

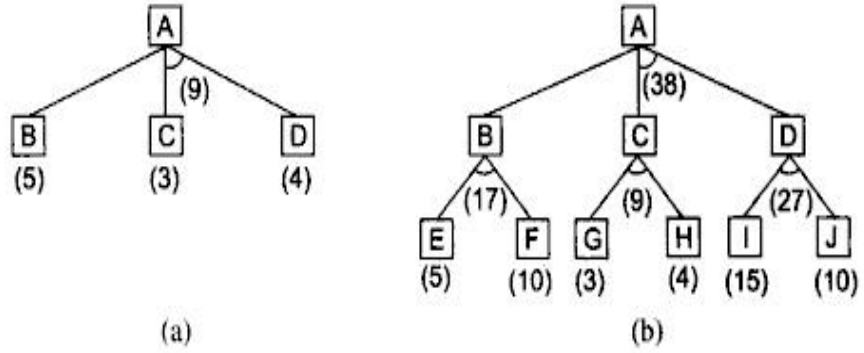


Fig. 3.7 *AND-OR Graphs*

In order to find solutions in an AND-OR graph, we need an algorithm similar to best-first search but with the ability to handle the AND arcs appropriately. This algorithm should find a path from the starting node of the graph to a set of nodes representing solution states. Notice that it may be necessary to get to more than one solution state since each arm of an AND arc must lead to its own solution node.

In order to describe an algorithm for searching an AND-OR graph, we need to exploit a value that we call *FUTILITY.* If the estimated cost of a solution becomes greater than the value of *FUTILITY,* then we abandon the search. *FUTILITY* should be chosen to correspond to a threshold such that any solution with a cosflibove it is too expensive to be practical, even if it could ever be found. Now we can state the algorithm.

Problem Reduction Example:-

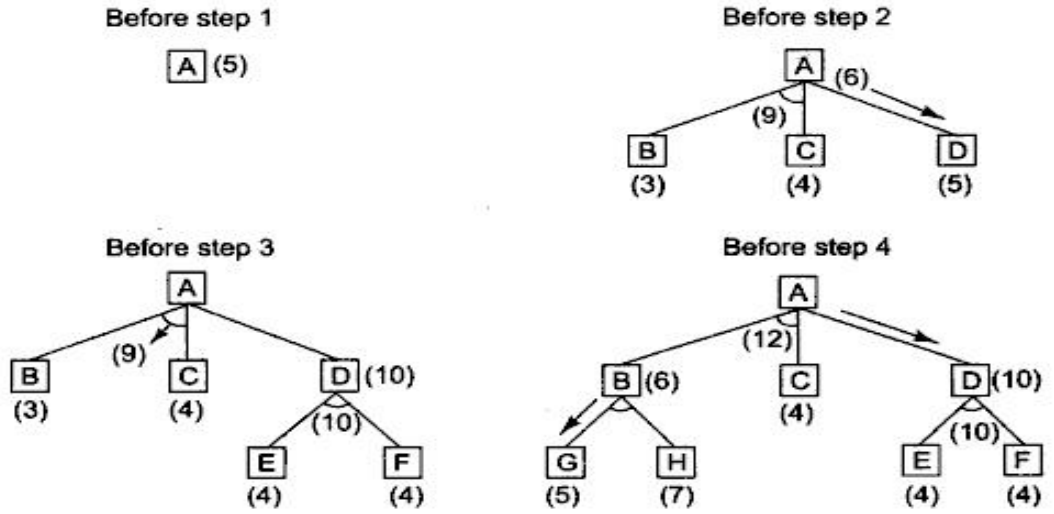


Fig. 3.8 *The Operation of Problem Reduction*

Algorithm: AO*

1. Let *GRAPH* consist only of the node representing the initial state. (Call this node *INIT.)* Compute *h′(INIT)*
2. Until *INIT* is labeled *SOLVED* or until *INIT's h′* value becomes greater than *FUTILITY,* repeat the following procedure:
   (a) Trace the labeled arcs from *INIT* and select for expansion one of the as yet unexpanded nodes that occurs on this path. Call the selected node *NODE.*
   (b) Generate the successors of *NODE.* If there are none, then assign *FUTILITY* as the *h′* value of *NODE.* This is equivalent to saying that *NODE* is not solvable. If there are successors, then for each one (called *SUCCESSOR)* that is not also an ancestor of *NODE* do the following:
      (i) Add *SUCCESSOR* to *GRAPH.*
      (ii) If *SUCCESSOR* is a terminal node, label it *SOLVED* and assign it an *h′* value of 0.
      (iii) If *SUCCESSOR* is not a terminal node, compute its *h′* value.
   (c) Propagate the newly discovered information up the graph by doing the following: Let *S* be a set of nodes that have been labeled *SOLVED* or whose *h′* values have been changed and so need to have values propagated back to their parents. Initialize 5 to *NODE.* Until *S* is empty, repeat the, following procedure:
      (i) If possible, select from *S* a node none of whose descendants in *GRAPH* occurs in *S.* If there is no such node, select any node from *S.* Call this node *CURRENT,* and remove it from *S.*
      (ii) Compute the cost of each of the arcs emerging from *CURRENT.* The cost of each arc is equal to the sum of the *h′* values of each of the nodes at the end of the arc plus whatever the cost of the arc itself is. Assign as *CURRENT'S* new *h′* value the minimum of the costs just computed for the arcs emerging from it.
      (iii) Mark the best path out of *CURRENT* by marking the arc that had the minimum cost as computed in the previous step.
      (iv) Mark *CURRENT SOLVED* if all of the nodes connected to it through the new labeled arc have been labeled *SOLVED.*
      (v) If *CURRENT* has been labeled *SOLVED* or if the cost of *CURRENT* was just changed, then its new status must be propagated back up the graph. So add all of the ancestors of *CURRENT* to *S.*

Q15. Explain Means-End Analysis with household robot example. Also explain difference table for it.

Ans: Means-ends Analysis :

Means-ends Analysis process centers around the detection of differences between the current state and the goal state. Once such a difference is isolated, an operator that can reduce the difference must be found.

If the operator cannot be applied to the current state, we set up a subproblem of getting to a state in which it can be applied.

The kind of backward chaining in which operators are selected and then subgoals are set up to establish the preconditions of the operators.

Household Robot domain. Give rules and Difference Table for it.

Solve the problem of getting from one place to another. Assume that the available operators are walk, drive, take bus, take a cab and fly.

Operator	*Preconditions*	*Results*
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)	none	at(robot, loc)
PICKUP(obj)	at(robot, obj)	holding(obj)
PUTDOWN(obj)	holding(obj)	¬holding(obj)
PLACE(obj1, obj2)	at(robot, obj2)^ holding(obj1)	on(obj1, obj2)

Fig. 3.15 *The Robot's Operators*

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

Fig. 3.16 *A Difference Table*

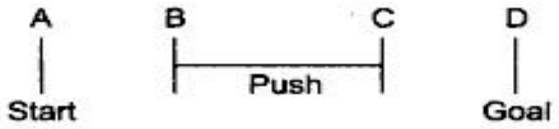


Fig. 3.17 *The Progress of the Means-Ends Analysis Method*

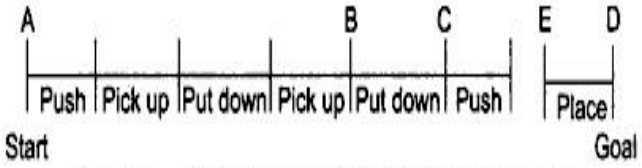


Fig. 3.18 *More Progress of the Means-Ends Method*

Operator	Preconditions	Results
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)	None	At(robot, loc)
PICKUP(Obj)	At(robot, obj)	Holding(obj)

PUTDOWN(obj)	Holding(obj)	¬holding(obj)
PLACE(Obj1, obj2)	At(robot, obj2)^ Holding(obj1)	On(obj1, obj2)

DIFFERENCE TABLE:-

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

Q16. Write a note on probability and Baye's theorem. [6M]

Ans: The formula is

$$P(A,B)=P(A|B)P(B)=P(B|A)P(A).$$

This is, in a way, the definition of conditional probability. Look at the first one:

$$P(A,B)=P(B|A)P(A).$$

There are many equivalent ways to interpret this.

For e.g: Let A be the event "I am hungry", and B the event "I go to a restaurant". Note that it's possible for me to be hungry and not go to a restaurant, and it's also possible for me to go to a restaurant without being hungry. But there is a correlation like this: The chance of me going to restaurant increases if I'm hungry. That means $P(B|A)>P(B)$. (This is only true in this example!) $P(B|A)$is called the conditional probability because it is conditioned on A. It is the probability of B happening given the knowledge that A happens.

The joint probability is $P(A,B)$, the chance of both A and B happening. If I know $P(A)$ and $P(B|A)$, I can compute $P(A,B)$ as follows. First, think about how probable A can happen. That is $P(A)$. And assuming A happens, how probable is it that B also happens? That is $P(B|A)$ by definition. Multiplying them together, I get the probability that both A and B happen.

- Specifying a probability for every atomic event is impractical
- We have already seen it can be easier to specify probability distributions by using (conditional) independence
- Bayesian networks allow us
  - to specify any distribution,

- to specify such distributions concisely if there is (conditional) independence, in a natural way

- Say the variables are $X_1, \ldots, X_n$
- $P(X_1, \ldots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1,X_2)\ldots P(X_n|X_1, \ldots, X_{n-1})$
- Can specify every component
- If every variable can take k values,
- $P(X_i|X_1, \ldots, X_{i-1})$ requires $(k-1)k^{i-1}$ values
- $\Sigma_{i=\{1,..,n\}}(k-1)k^{i-1} = \Sigma_{i=\{1,..,n\}}k^i - k^{i-1} = k^n - 1$
- Same as specifying probabilities of all atomic events – of course, because we can specify any distribution!

Q17. Explain certainty factors and explain why they are needed. [7M]

Ans: The certainty-factor (CF) model is a commonly used method for managing uncertainty in rule-based systems. We review the history and mechanics of the CF model, and delineate precisely its theoretical and practical limitations. In addition, we examine the belief network, a representation that is similar to the CF model but that is grounded firmly in probability theory. We show that the belief-network representation overcomes many of the limitations of the CF model, and provides a promising approach to the practical construction of expert systems. The certainty-factor (CF) model is a commonly used method for managing uncertainty in rule-based systems. We review the history and mechanics of the CF model, and delineate precisely its theoretical and practical limitations. In addition, we examine the belief network, a representation that is similar to the CF model but that is grounded firmly in probability theory. We show that the belief-network representation overcomes many of the limitations of the CF model, and provides a promising approach to the practical construction of expert systems. The certainty-factor (CF) model is a commonly used method for managing uncertainty in rule-based systems. We review the history and mechanics of the CF model, and delineate precisely its theoretical and practical limitations. In addition, we examine the belief network, a representation that is similar to the CF model but that is grounded firmly in probability theory. We show that the belief-network representation overcomes many of the limitations of the CF model, and provides a promising approach to the practical construction of expert systems.

The belief network is such a language. A generalization of the belief network in which we can represent decisions and the preferences of a decision maker. The belief network is a directed acyclic graph. The nodes in the graph correspond to uncertain variables relevant to the problem. For Mr. Holmes, each uncertain variable represents a proposition and that proposition's negation. For example, node b in Figure 4 represents the propositions BURGLARY and NOT BURGLARY (denoted b+ and b−, respectively). In general, an uncertain variable can represent an arbitrary set of mutually exclusive and exhaustive propositions; we call each proposition an instance of the variable. In the remainder of the discussion, we make no distinction between the variable x and the node x that represents that variable. Each variable in a belief network is associated with a set of probability distributions. In the Bayesian tradition, these distributions encode the knowledge provider's beliefs about the relationships among the variables.

Q18. What is fuzzy logic? Write an example and explain it. [9M]

Ans: Fuzzy logic is a form of many-valued logic; it deals with reasoning that is approximate rather than fixed and exact. Compared to traditional binary sets (where variables may take ontrue or false values), fuzzy logic variables may have a truth value that ranges in degree between 0 and 1. Fuzzy logic has been extended to handle the concept of partial truth, where the truth value may range between completely true and completely false.

Fuzzy logic has been around since the mid 1960s; however, it was not until the 70s that a practical application was demonstrated. Since that time the Japanese have traditionally been the largest producer of fuzzy logic applications. Fuzzy logic has appeared in cameras, washing machines, and even in stock trading applications. In the last decade the United States has started to catch on to the use of fuzzy logic. There are many applications that use fuzzy logic, but fail to tell us of its use. Probably the biggest reason is that the term "fuzzy logic" may have a negative connotation. Fuzzy logic can be applied to non-engineering applications as illustrated in the stock trading application. It has also been used in medical diagnosis systems and in handwriting recognition applications. In fact a fuzzy logic system can be applied to almost any type of system that has inputs and outputs.

Fuzzy logic systems are well suited to nonlinear systems and systems that have multiple inputs and multiple outputs. Any reasonable number of inputs and outputs can be accommodated. Fuzzy logic also works well when the system cannot be modeled easily by conventional means.

Many engineers are afraid to dive into fuzzy logic due to a lack of understanding. Fuzzy logic does not have to be hard to understand, even though the math behind it can be intimidating, especially to those of us who have not been in a math class for many years.

Binary logic is either 1 or 0. Fuzzy logic is a continuum of values between 0 and 1. This may also be thought of as 0% to 100%. An example is the variable YOUNG. We may say that age 5 is 100% YOUNG, 18 is 50% YOUNG, and 30 is 0% YOUNG. In the binary world everything below 18 would be 100% YOUNG, and everything above would be 0% YOUNG.

The design of a fuzzy logic system starts with a set of membership functions for each input and a set for each output. A set of rules is then applied to the membership functions to yield a "crisp" output value.

For this process control explanation of fuzzy logic, TEMPERATURE is the input and FAN SPEED is the output. Create a set of membership functions for each input. A membership function is simply a graphical representation of the fuzzy variable sets. For this example, use three fuzzy sets, COLD, WARM, and HOT. We will then create a membership function for each of three sets of temperature as shown in the cold-normal-hot graphic.

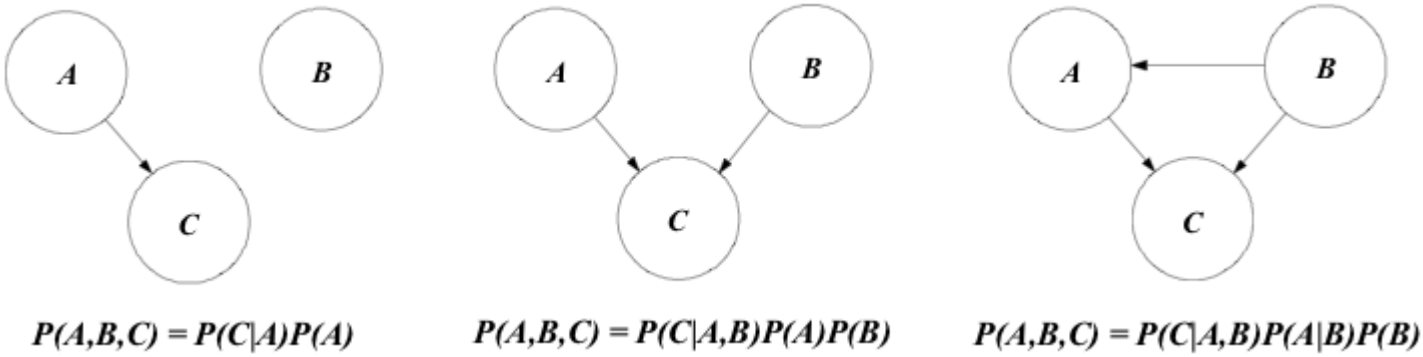
Q20. **Explain Baysian Network in brief.** **[6M]**

Ans: Bayesian networks provide a means of parsimoniously expressing joint probability distributions over many interrelated hypotheses. A Bayesian network consists of a directed acyclic graph (DAG) and a set of local distributions. Each node in the graph represents a random variable. A random variable denotes an attribute, feature, or hypothesis about which we may be uncertain. Each random variable has a set of mutually exclusive and collectively exhaustive possible values. That is, exactly one of the possible values is or will be the actual value, and we are uncertain about which one it is. The graph represents direct qualitative dependence relationships; the local distributions represent quantitative information about the strength of those dependencies. The graph and the local distributions together represent a joint distribution over the random variables denoted by the nodes of the graph.

Bayesian networks have been successfully applied to create consistent probabilistic representations of uncertain knowledge in diverse fields such as medical diagnosis (Spiegelhalter *et al.*, 1989), image recognition (Booker & Hota, 1986), language understanding (Charniak & Goldman, 1989a, 1989b), search algorithms (Hansson & Mayer,1989), and many others. Heckerman *et. al.* (1995b) provides a detailed list of recent applications of Bayesian Networks.

Prof. Neha Mogre

One of the most important features of Bayesian networks is the fact that they provide an elegant mathematical structure for modeling complicated relationships among random variables while keeping a relatively simple visualization of these relationships. The figure below gives three simple examples of qualitatively different probability relationships among three random variables.



$P(A,B,C) = P(C|A)P(A)$ $P(A,B,C) = P(C|A,B)P(A)P(B)$ $P(A,B,C) = P(C|A,B)P(A|B)P(B)$

As a means for realizing the communication power of this representation, one could compare two hypothetical scenarios in which a domain expert with little background in probability tries to interpret what is represented in the figure above. Initially, suppose that she is allowed to look only to the written equations below the pictures. In this case, we believe that she will have to think at least twice before making any conclusion on the relationships among events *A*, *B*, and *C*. On the other hand, if she is allowed to look only to the pictures, it seems fair to say that she will immediately perceive that in the leftmost picture, for example, event *B* is independent of events *A* and *C*, and event *C*depends on event *A*. Also, simply comparing the pictures would allow her to see that, in the center picture, *A* is now dependent on *B*, and that in the rightmost picture *B* influences both *A* and *C*. Advantages of easily interpretable graphical representation become more apparent as the number of hypothesis and the complexity of the problem increases. Pearl's algorithm performs exact Bayesian updating, but only for singly connected networks. Subsequently, general Bayesian updating algorithms have been developed. One of the most commonly applied is the Junction Tree algorithm (Lauritzen & Spiegelhalter, 1988). Neapolitan (2003) provides a discussion on many Bayesian propagation algorithms. Although Cooper (1987) showed that exact belief propagation in Bayesian Networks can be NP-Hard, exact computation is practical for many problems of practical interest. Some complex applications are too challenging for exact inference, and require approximate solutions (Dagum & Luby, 1993). Many computationally efficient inference algorithms have been developed, such as probabilistic logic sampling (Henrion, 1988), likelihood weighting (Fung & Chang, 1989; Shachter & Peot, 1990), backward sampling (Fung & del Favero, 1994), Adaptive Importance Sampling (Cheng & Druzdzel, 2000), and Approximate Posterior Importance Sampling (Druzdzel & Yuan, 2003).

Those algorithms allow the impact of evidence about one node to propagate to other nodes in multiply-connected trees, making Bayesian Networks a reliable engine for probabilistic inference. The prospective reader will find comprehensive coverage of Bayesian Networks in a large and growing literature on this subject, such as Pearl (1988), Neapolitan (1990, 2003), Oliver & Smith (1990), Charniak (1991), Jensen (1996, 2001), or Korb & Nicholson (2003).

Limitations of Probabilistic Reasoning with Bayesian Networks:

Prof. Neha Mogre

Bayesian Networks have received praise for being a powerful tool for performing probabilistic inference, but they do have some limitations that impede their application to complex problems. As the technique grew in popularity, Bayesian Network's limitations became increasingly apparent. One of the most important limitations for it to be applied in the context of PR-OWL is the fact that, although a powerful tool, BNs are not expressive enough for many real-world applications. More specifically, Bayesian Networks assume a simple attribute-value representation – that is, each problem instance involves reasoning about the same fixed number of attributes, with only the evidence values changing from problem instance to problem instance.This type of representation is inadequate for many problems of practical importance. Many domains require reasoning about varying numbers of related entities of different types, where the numbers, types and relationships among entities usually cannot be specified in advance and may have uncertainty in their own definitions. As will be demonstrated below, Bayesian networks are insufficiently expressive for such problems.

Q21. Describe Truth Maintenance system with example. [7M]

Ans:

Truth maintenance systems (TMS) were introduced more than ten years ago, but recently there is an explosion of interest in them and their possible applications in different areas. In this paper we discuss truth maintenance from three perspectives:

(i) Truth maintenance as a *data base management facility*, which was in fact the original intention of the TMS.

(ii) Truth maintenance as an *inference facility*, which provides a way to extend the role of the TMS in solving problems.

(iii) Truth maintenance as a *verification facility*, which illustrates a new and promising application of TMSs in the area of expert systems design. Truth Maintenance Systems (TMS), also called Reason Maintenance Systems, are used within Problem Solving Systems, in conjunction with Inference Engines (IE) such as rule-based inference systems, to manage as a Dependency Network the inference engine's beliefs in given sentences.

The Truth Maintenance System (TMS) is a problem solver subsystem for performing these functions by recording and maintaining the reasons for program beliefs. Such recorded reasons are useful in constructing explanations of program actions in guiding the course of action of a problem solver.

(1) the representations and structure of the TMS

(2) the mechanisms used to revise the current set of beliefs

(3) how dependency-directed backtracking changes the current set of assumptions

(4) techniques for summarizing explanations of beliefs

(5) how to organize problem solvers into "dialectically arguing" modules

(6) how to revise models of the belief systems of others

(7) methods for embedding control structures in patterns of assumptions.

Q22. Explain non monotonic reasoning system in brief with suitable example. [8M]

Ans:-

Non-Monotonic Reasoning:

Predicate logic and the inferences we perform on it is an example of *monotonic* reasoning. In monotonic reasoning if we enlarge at set of axioms we cannot retract any existing assertions or axioms.

Humans do not adhere to this monotonic structure when reasoning:

(i) We need to jump to conclusions in order to plan and, more basically, survive.

(ii) We cannot anticipate all possible outcomes of our plan.

(iii)We must make assumptions about things we do not specifically know about.

Default reasoning:- This is a very common from of non-monotonic reasoning. Here we want to draw conclusions based on what is most likely to be true. We have already seen examples of this and possible ways to represent this knowledge.

We will discuss two approaches to do this:

- Non-Monotonic logic.
- Default logic.

DO NOT get confused about the label *Non-Monotonic* and *Default* being applied to reasoning and a particular logic. Non-Monotonic reasoning is generic descriptions of a class of reasoning. Non-Monotonic logic is a specific theory. The same goes for Default reasoning and Default logic.

Non-Monotonic Logic

This is basically an extension of first-order predicate logic to include a *modal* operator, *M*. The purpose of this is to allow for consistency.

For example: $\forall x$: plays_instrument(x) $\wedge^{M}$ improvises(x) $\rightarrow$ jazz_musician(x)

states that for all *x* is *x* plays an instrument and if the fact that *x* can improvise is consistent with all other knowledge then we can conclude that *x* is a jazz musician.

How do we define *consistency*?

One common solution (consistent with PROLOG notation) is to show that fact *P* is true attempt to prove $\neg P$. If we fail we may say that *P* is consistent (since $\neg P$is false).

However consider the famous set of assertions relating to President Nixon.

$\forall x$: Republican(x) $\wedge^{M\neg}$ Pacifist(x) $\rightarrow$ $\neg$Pacifist(x)

$\forall x$: Quaker(x) $\wedge^{M}$ Pacifist(x) $\rightarrow$ Pacifist(x)

Now this states that Quakers tend to be pacifists and Republicans tend not to be.

BUT Nixon was both a Quaker and a Republican so we could assert:

Quaker(Nixon)

Republican(Nixon)

This now leads to our total knowledge becoming inconsistent.

Default Logic

Default logic introduces a new inference rule:

A:B/C which states if A is deducible and it is consistent to assume B then conclude C.

Now this is similar to Non-monotonic logic but there are some distinctions:

- New inference rules are used for computing the set of plausible extensions. So in the Nixon example above Default logic can support both assertions since is does not say anything about how choose between them -- it will depend on the inference being made.
- In Default logic any nonmonotonic expressions are rules of inference rather than expressions.

Circumscription

Circumscription is a rule of conjecture that allows you to jump to the conclusion that the objects you can show that posses a certain property, *p*, are in fact all the objects that posses that property.

Circumscription can also cope with default reasoning.

Suppose we know: bird(tweety)

$\forall x$: penguin(x) $\rightarrow$bird(x)

$\forall x$: penguin(x) $\rightarrow$ $\neg$flies(x)

and we wish to add the fact that *typically, birds fly*.

In circumscription this phrase would be stated as:

A bird will fly if it is not abnormal and can thus be represented by:

$\forall x$: bird(x) $\wedge \neg$abnormal(x) $\rightarrow$flies(x).

However, this is not sufficient,we cannot conclude

flies(tweety)

since we cannot prove

$\neg$abnormal(tweety).

This is where we apply circumscription and, in this case, we will assume that those things that are shown to be abnormal are the only things to be abnormal

Thus we can rewrite our *default rule* as:

$\forall x$: bird(x) $\wedge \neg$ flies(x) $\rightarrow$ abnormal(x)

and add the following

$\forall x$: $\neg$abnormal(x)

since there is nothing that cannot be shown to be abnormal.

If we now add the fact:

penguin(tweety)

Clearly we can prove

abnormal(tweety).

If we circumscribe abnormal now we would add the sentence,

a penguin (tweety) is the abnormal thing:

$\forall x$: abnormal(x) $\rightarrow$ penguin(x).

Q23. What is learning? Explain in brief different types of learning. [6M]
Ans:-

Learning is an important area in AI, perhaps more so than planning.

- Problems are hard -- harder than planning.
- Recognised Solutions are not as common as planning.
- A goal of AI is to enable computers that can be taught rather than programmed.

Learning is a an area of AI that focusses on processes of self-improvement.

Information processes that improve their performance or enlarge their knowledge bases are said to learn.

Why is it hard?

- Intelligence implies that an organism or machine must be able to adapt to new situations.
- It must be able to learn to do new things.
- This requires knowledge acquisition, inference, updating/refinement of knowledge base, acquisition of heuristics, applying faster searches, etc.
- How can we learn?
- Many approaches have been taken to attempt to provide a machine with learning capabilities. This is because learning tasks cover a wide range of phenomena.
- Listed below are a few examples of how one may learn. We will look at these in detail shortly

- **Skill refinement :** one can learn by practicing, e.g playing the piano.
- **Knowledge acquisition :** one can learn by experience and by storing the experience in a knowledge base. One basic example of this type is rote learning.
- Taking advice
- -- Similar to rote learning although the knowledge that is input may need to be transformed (or operationalised) in order to be used effectively.
- Problem Solving
- -- if we solve a problem one may learn from this experience. The next time we see a similar problem we can solve it more efficiently. This does not usually involve gathering new knowledge but may involve reorganisation of data or remembering how to achieve to solution.
- Induction
- -- One can learn from examples. Humans often classify things in the world without knowing explicit rules. Usually involves a teacher or trainer to aid the classification.
- Discovery
- -- Here one learns knowledge without the aid of a teacher.
- Analogy
- -- If a system can recognise similarities in information already stored then it may be able to transfer some knowledge to improve to solution of the task in hand.

Q24. Give general learning model, also give the role of each component in learning model. [7M]

Ans:-

General Learning Model: - AS noted earlier, learning can be accomplished using a number of different methods, such as by memorization facts, by being told, or by studying examples like problem solution. Learning requires that new knowledge structures be created from some form of input stimulus. This new knowledge must then be assimilated into a knowledge base and be tested in some way for its utility. Testing means that the knowledge should be used in performance of some task from which meaningful feedback can be obtained, where the feedback provides some measure of the accuracy and usefulness of the newly acquired knowledge.

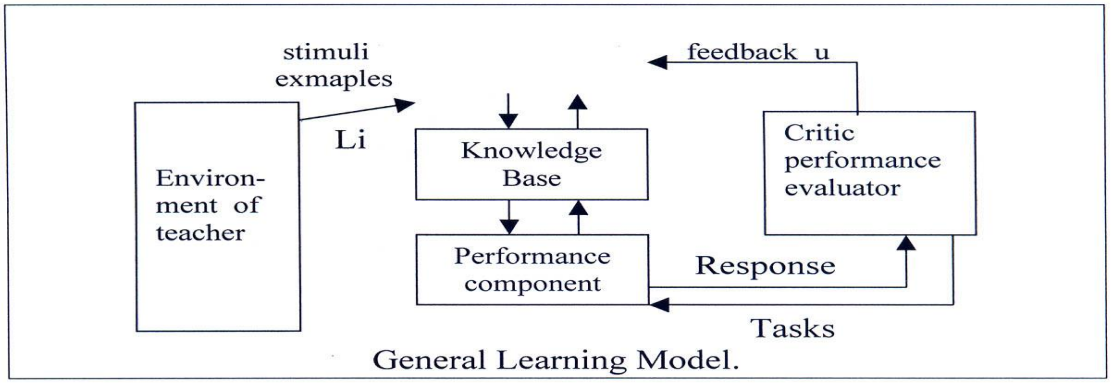


Fig.General Learning Model.

General learning model is depicted in figure where the environment has been included as a part of the overall learner system. The environment may be regarded as either a form of nature which produces random stimuli or as a more organized training source such as a teacher which provides carefully selected training examples for the learner component. The actual form of environment used will depend on the particular learning

paradigm. In any case, some representation language must be assumed for communication between the environment and the learner. The language may be the same representation scheme as that used in the knowledge base (such as a form of predicate calculus). When they are chosen to be the same, we say the single representation trick is being used. This usually results in a simpler implementation since it is not necessary to transform between two or more different representations.

For some systems the environment may be a user working at a keyboard . Other systems will use program modules to simulate a particular environment. In even more realistic cases the system will have real physical sensors which interface with some world environment.

Inputs to the learner component may be physical stimuli of some type or descriptive , symbolic training examples. The information conveyed to the learner component is used to create and modify knowledge structures in the knowledge base. This same knowledge is used by the performance component to carry out some tasks, such as solving a problem playing a game, or classifying instances of some concept.

given a task, the performance component produces a response describing its action in performing the task. The critic module then evaluates this response relative to an optimal response.

Feedback , indicating whether or not the performance was acceptable , is then sent by the critic module to the learner component for its subsequent use in modifying the structures in the knowledge base. If proper learning was accomplished, the system's performance will have improved with the changes made to the knowledge base.

The cycle described above may be repeated a number of times until the performance of the system has reached some acceptable level, until a known learning goal has been reached, or until changes ceases to occur in the knowledge base after some chosen number of training examples have been observed.

There are several important factors which influence a system's ability to learn in addition to the form of representation used. They include the types of training provided, the form and extent of any initial background knowledge , the type of feedback provided, and the learning algorithms used.

The type of training used in a system can have a strong effect on performance, much the same as it does for humans. Training may consist of randomly selected instance or examples that have been carefully selected and ordered for presentation. The instances may be positive examples of some concept or task a being learned, they may be negative, or they may be mixture of both positive and negative. The instances may be well focused using only relevant information, or they may contain a variety of facts and details including irrelevant data.

There are Many forms of learning can be characterized as a search through a space of possible hypotheses or solutions. To make learning more efficient. It is necessary to constrain this search process or reduce the search space. One method of achieving this is through the use of background knowledge which can be used to constrain the search space or exercise control operations which limit the search process.

Feedback is essential to the learner component since otherwise it would never know if the knowledge structures in the knowledge base were improving or if they were adequate for the performance of the given tasks. The feedback may be a simple yes or no type of evaluation, or it may contain more useful information describing why a particular action was good or bad. Also , the feedback may be completely reliable, providing

an accurate assessment of the performance or it may contain noise, that is the feedback may actually be incorrect some of the time. Intuitively , the feedback must be accurate more than 50% of the time; otherwise the system carries useful information, the learner should also to build up a useful corpus of knowledge quickly. On the other hand, if the feedback is noisy or unreliable, the learning process may be very slow and the resultant knowledge incorrect.

Q25. Write a short notes on Rote learning. [6M]

Ans:- Rote Learning:- Rote Learning is basically memorisation.

- Saving knowledge so it can be used again.
- Retrieval is the only problem.
- No repeated computation, inference or query is necessary.

A simple example of rote learning is caching

- Store computed values (or large piece of data)
- Recall this information when required by computation.
- Significant time savings can be achieved.
- Many AI programs (as well as more general ones) have used caching very effectively.

Memorisation is a key necessity for learning:

- It is a basic necessity for any intelligent program -- is it a separate learning process?
- Memorisation can be a complex subject -- how best to store knowledge?

Samuel's Checkers program employed rote learning

Q26. Write a short notes on Learning by taking advice. [6M]

Ans:- The idea of advice taking in AI based learning was proposed as early as 1958 (McCarthy). However very few attempts were made in creating such systems until the late 1970s. Expert systems providing a major impetus in this area.

There are two basic approaches to advice taking:

- Take high level, abstract advice and convert it into rules that can guide performance elements of the system. Automate all aspects of advice taking
- Develop sophisticated tools such as knowledge base editors and debugging. These are used to aid an expert to translate his expertise into detailed rules. Here the expert is an integral part of the learning system. Such tools are important in expert systems area of AI.

Q27. Explain First Order Logic. [7M]

Ans: So far we studied propositional logic

Some English statements are hard to model in propositional logic:

- "If your roommate is wet because of rain, your roommate must not be carrying any umbrella"

RoommateWetBecauseOfRain=>(NOT(RoommateCarryingUmbrella0)AND NOT(RoommateCarryingUmbrella1) AND NOT(RoommateCarryingUmbrella2) AND …)

- No notion of objects
- No notion of relations among objects
- RoommateCarryingUmbrella0 is instructive to us, suggesting
  - there is an object we call Roommate,
  - there is an object we call Umbrella0,
  - there is a relationship Carrying between these two objects
- Formally, none of this meaning is there
  - Might as well have replaced RoommateCarryingUmbrella0 by P
- Objects: can give these names such as Umbrella0, Person0, John, Earth, …
- Relations: Carrying(., .), IsAnUmbrella(.)
  - Carrying(Person0, Umbrella0), IsUmbrella(Umbrella0)
  - Relations with one object = unary relations = properties
- Functions: Roommate(.)
  - Roommate(Person0)
- Equality: Roommate(Person0) = Person1

Q28. First order Logic Problem.

Ans: Given the following sentences, write them in first-order logic.
(a) John likes all kinds of foods.
-> ∀x.food(x) → likes(John, x)
(b) Apples are food.
-> food(Apple)
(c) Chicken is food.
-> food(Chicken)
(d) Anything anyone eats and remains alive is food.
-> ∀x, y.(eats(x, y) ∧ alive(x)) → food(y)
(e) Bill eats peanuts and is alive.
-> eats(Bill, P eanut) ∧ alive(Bill)
(f) Sue eats everything that Bill eats.
-> ∀x:eats(Bill, x) → eats(Sue, x)

Convert the FOL sentences into Clause Form.
(a) ¬food(x) ∨ likes(John, x)
(b) food(Apple)
(c) food(Chicken)
(d) ¬eats(x, y) ∨ ¬alive(x) ∨ food(y)
(e) eats(Bill, Peanut)
(f) alive(Bill)
(g) ¬eats(Bill, x) ∨ eats(Sue, x)
C. Prove, by resolution refutation (contradiction), that John likes peanuts.
Negate conclusion.
¬likes(John, Peanuts)
Substitute Bill for x and Peanuts for y into (d)
¬eats(Bill, Peanuts) ∨ ¬alive(Bill) ∨ food(Peanuts)
Resolve (e), (f) and above.
food(Peanuts)
Resolve (a) and above using substitution x for John and y for Peanuts.

likes(John, Peanuts)
Contradiction.

Q29. Explain Components of Expert system in detail. Also explain Characteristics of Expert System. [7M]

Ans: Expert system is an artificial intelligence program that has expert-level knowledge about a particular domain and knows how to use its knowledge to respond properly.

- Domain refers to the area within which the task is being performed.
- Ideally the expert systems should substitute a human expert.
- *Edward Feigenbaum* of Stanford University has defined expert system as "an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solutions."
- It is a branch of artificial intelligence introduced by researchers in the Stanford Heuristic Programming Project.
- The *expert systems* is a branch of AI designed to work within a particular domain.
- As an expert is a person who can solve a problem with the domain knowledge in hands it should be able to solve problems at the level of a human expert.
- The source of knowledge may come from a human expert and/or from books, magazines and internet.
- As knowledge play a key role in the functioning of expert systems they are also known as knowledge-based systems and knowledge-based expert systems.
- The expert's knowledge about solving the given specific problems is called knowledge domain of the expert.

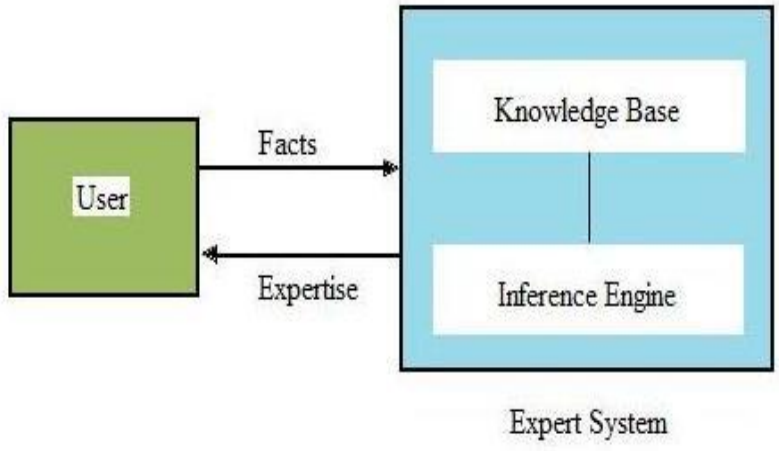


Fig. Components of Expert System

Basic Concept of an Expert System Function

- The expert system consists of two major components: knowledge base and inference engine.
- **Knowledge base** contains the domain knowledge which is used by the inference engine to draw conclusions.
- The inference engine is the generic control mechanism that applies the axiomatic knowledge to the task-specific data to arrive at some conclusion.
- When a user supplies facts or relevant information of query to the expert system he receives advice or expertise in response. That is given the facts it uses the inference engine which in turn uses the knowledge base to infer the solution.

Characteristics of Expert System:

- **High performance:** They should perform at the level of a human expert.

- **Adequate response time:** They should have the ability to respond in a reasonable amount of time. Time is crucial especially for real time systems.
- **Reliability:** They must be reliable and should not crash.
- **Understandable:** They should not be a black box instead it should be able explain the steps of the reasoning process. It should justify its conclusions in the same way a human expert explains why he arrived at particular conclusion.

Q30. What are Advantages of Expert System. [6M]

Ans: Availability: Expert systems are available easily due to mass production software.

- **Cheaper:** The cost of providing expertise is not expensive.
- **Reduced danger:** They can be used in any risky environments where humans cannot work with.
- **Permanence:** The knowledge will last long indefinitely.
- **Multiple expertise:** It can be designed to have knowledge of many experts.
- **Explanation:** They are capable of explaining in detail the reasoning that led to a conclusion.
- **Fast response:** They can respond at great speed due to the inherent advantages of computers over humans.
- **Unemotional and response at all times:** Unlike humans, they do not get tense, fatigue or panic and work steadily during emergency situations.
- Improved Decision Quality
- Increased Output and Productivity
- Decreased Decision Making Time
- Increased Process(es) and Product Quality
- Capture Scarce Expertise
- Can Work with Incomplete or Uncertain Information
- Enhancement of Problem Solving and Decision Making
- Improved Decision Making Processes
- Knowledge Transfer to Remote Locations
- Enhancement of Other MIS

Q31. What is Rule based expert system design. [6M]

Ans:

Rules are the popular paradigm for representing knowledge. A rule based expert system is one whose knowledge base contains the domain knowledge coded in the form of rules.

ELEMENTS OF A RULE BASED EXPERT SYSTEM

A rule based expert system consists of the following components:

USER INTERFACE

This is a mechanism to support communication between and the system. The user interface may be a simple text-oriented display or a sophisticated, high resolution display. It is determined at the time of designing the system. Nowadays graphical user interfaces are very common for their user-friendliness.

EXPLANATIONFACILITY

It explains the user about the reasoning process of the system. By keeping track of the rules that are fired, an

explanation facility presents a chain of reasoning that led to a certain conclusion. So explanation facility is also called justifier. This feature makes a huge difference between expert systems and other conventional systems. almost all the commercial expert system shells do trace based explanation, that is, explaining the inferencing on a specific input data set. Some systems explain the knowledge base itself, and some explain the control strategy as well.

WORKING MEMORY

This is a database used to store collection of facts which will later be used by the rules. More effort may go into the design and implementation of the user interface than in the expert system knowledge base. Working memory is used by the inference engine to get facts and match them against the rules. The facts may be added to the working memory by applying some rules.

INFERENCE ENGINE

As the name implies the inference engine makes inferences. It decides which rules are satisfied by the facts, prioritizes them, and executes the rule with the highest priority. There are two types of inference: forward chaining and backward chaining. Forward chaining is reasoning from facts to the conclusion while backward chaining is from hypothesis to the facts that support this hypothesis. Whether an inference engine performs forward chining or backward chaining entirely depends on the design which in turn depends on the type of problem. Some of the systems that do forward chaining are OPS5 and CLIPS. EMYCIN one of the most popular systems performs backward chining. Some systems, ART and KEE, for example, offer both the techniques. Forward chaining is best suited for prognosis, monitoring and control. Backward chaining is generally used for diagnostic problems. Inference engine operates in cycles, executing a group of tasks until certain criteria causes that halt the execution. The taks to be done repeatedly are conflict resolution, act, match and check for halt. Multiple rules may be activated and put on the agenda during one cycle.

AGENDA

Inference engine prepares a priotorized list of rules called agenda. The rules in the list must be satisfied by the facts in the working memory. When the inference engine notices a fact that satisfies the pattern in the condition part of the rule then it adds the rule to the agenda. If a rule has mulitple patterns then all of its patterns must be satisfied to get place in the agend. In a condition a>b and b>c, for example, both a>b and b>c must be satisfied. A rule whose patterns are satisfied is said to be activated or instantiated. When there is more than one activated rule in the agenda then the inference engine has to select one rule, based on priority or on other factors, for firing. Rule based expert systems are built using refraction to prevent trivial loops. That is a rule which is fired on a fact will not be fired again and again on the same fact. To implement this feature OPS5 uses a unique identifier called timetag. Then-part of the rule contains actions to be executed when the rule fires. Some of the actions are addition of facts to the working memory, removal of facts from the working memory and printing results. Agenda conflict occurs when different activations have the same priority. Agenda conflict is tackled by strategies such as first come first execute, assigning default priority, and so on.

KNOWLEDGE ACQUISITION FACILITY

This allows the user to enter knowledge in the system thereby avoiding the need of knowledge engineer explicitly code the knowledge. It is an optional feature on many expert systems. Simple rules can be created using rule induction. In rule based expert systems, knowledge base is also called production memory as rules in the form of if-then are called productions.

Prof. Neha Mogre

ADVANTAGES OF RULE BASED EXPERT SYSTEMS

Modular nature: This allows encapsulating knowledge and expansion of the expert system done in a a easy way.

Explanation facilities: Rules make it easy to build explanation facilities. By keeping track of the rules that are fired, an explanation facility can present a chain of reasoning that led to a certain conclusion.

Similarity to the human cognitive process: Newel and Simon have showed that rules are the natural way of modeling how humans solve problems. Rules make it easy to explain the structure of knowledge to the experts.

Q32. Explain Natural language processing (NLP). [7M]

Ans: Natural language processing (NLP) is the ability of a computer program to understand human speech as it is spoken.

- NLP is a component of artificial intelligence (AI).
- The development of NLP applications is challenging because computers traditionally require humans to "speak" to them in a programming language that is precise, unambiguous and highly structured or, perhaps through a limited number of clearly-enunciated voice commands.
- Human speech, however, is not always precise -- it is often ambiguous and the linguistic structure can depend on many complex variables, including slang, regional dialects and social context.
- Current approaches to NLP are based on machine learning, a type of artificial intelligence that examines and uses patterns in data to improve a program's own understanding.
- Most of the research being done on natural language processing revolves around search, especially enterprise search.
- Common NLP tasks in software programs today include:
- Sentence segmentation, part-of-speech tagging and parsing.
- Deep analytics.
- Named entity extraction.
- Co-reference resolution.

  The advantage of natural language processing can be seen when considering the following two statements: "Cloud computing insurance should be part of every service level agreement" and "A good SLA ensures an easier night's sleep -- even in the cloud."
- If you use national language processing for search, the program will recognise that *cloud computing* is an entity, that *cloud* is an abbreviated form of cloud computing and that *SLA* is an industry acronym for service level agreement.
- The ultimate goal of NLP is to do away with computer programming languages altogether. Instead of specialized languages such as Java or Ruby or C, there would only be "human."

Q33. Write a short note on basic parsing techniques. [7M]

Ans: The result is a *parse tree*. A parse tree is a structure where each node is a symbol from the grammar. The root node is the *starting nonterminal*, the intermediate nodes are nonterminals, the leaf nodes are terminals.

- "Sentence" is the *starting nonterminal*.
- There are two classes of parsing algorithms

- *top-down parsers*: start with the starting symbol and try to derive the complete sentence
- *bottom-up parsers*: start with the complete sentence and attempt to find a series of reductions to derive the start symbol

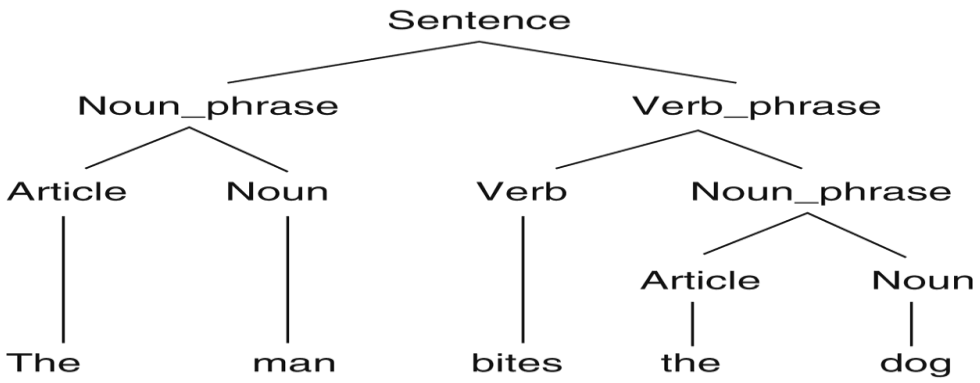


Fig. Parse Tree

- Search for the correct derivation
- If a wrong choice is made, the parser needs to backtrack
- *Recursive descent parsers* maintain backtrack pointers
- *Look-ahead techniques* help determine the proper rule to apply

We'll study *transition network parsers* (and *augmented transition networks*)

1. Each time the function parse is called with a terminal symbol as argument and that terminal matches the next symbol of input, it returns a tree consisting of a single leaf node labeled with that symbol.

2. When parse is called with a nonterminal, N, it calls transition. If transition succeeds, it returns an ordered set of subtrees. Parse combines these into a tree whose root is N and whose children are the subtrees returned by transition.

Q34. Explain Knowledge acquisition process in brief. [6M]

Ans: Acquiring and validating a large groups of consistent, correlated knowledge is not a trivial problem . This has give the acquisition process an especially important role in the design and implementation of these systems. Consequently, effective acquisition methods have become one of the principal challenges for the AI researches.

The goals of this branch of AI are the discovery and development of efficient, cost effective methods of acquisition. Some important progress has recently been made in this area with the development of sophisticated editors and some general concepts related to acquisition and learning.

Definition:- Knowledge acquisition is the process of adding new knowledge to a knowledge base and refining or otherwise improving knowledge that was previously acquired. Acquisition is usually associated with some purpose such as expanding the capabilities of a system or improving its performance at some specified task. It is goal oriented creation and refinement of knowledge. It may consist of facts, rules, concepts, procedures, heuristics, formulas, relationships, statistics or other useful information.

Q35. Describe different forms of knowledge required for Natural Language Understanding. [13M]

Ans:

- PHONETIC AND PHONOLOGICAL KNOWLEDGE
- MORPHOLOGICAL KNOWLEDGE
- SYNTACTIC KNOWLEDGE
- SEMANTIC KNOWLEDGE
- PRAGMATIC KNOWLEDGE
- DISCOURSE KNOWLEDGE
- WORLD KNOWLEDGE

1.PHONETIC AND PHONOLOGICAL KNOWLEDGE: Phonetics is the study of language at the level of sounds while phonology is the study of combination of sounds into organized units of speech, the formation of syllables and larger units. Phonetic and phonological knowledge are essential for speech based systems as they deal with how words are related to the sounds that realize them.

2.MORPHOLOGICAL KNOWLEDGE: Morphology concerns word formation. It is a study of the patterns of formation of words by the combination of sounds into minimal distinctive units of meaning called mophemes. Morphological knowledge concerns how words are constructed from morphemes.

3.SYNTACTIC KNOWLEDGE: Syntax is the level at which we study how words combine to form phrases, phrases combine to form clauses and clauses join to make sentences.

Syntactic analysis concerns sentence formation. It deals with how words can be put together to form correct sentences.

It also determines what structural role each word plays in the sentence and what phrases are subparts of what other phrases.

4.SEMANTIC KNOWLEDGE: It concerns meanings of the words and sentences. This is the study of context independent meaning that is the meaning a sentence has, no matter in which context it is used. Defining the meaning of a sentence is very difficult due to the ambiguities involved

5.PRAGMATIC KNOWLEDGE: Pragmatics is the extension of the meaning or semantics. Pragmatics deals with the contextual aspects of meaning in particular situations. It concerns how sentences are used in different situations and how use affects the interpretation of the sentence.

6.DISCOURSE KNOWLEDGE: Discourse concerns connected sentences. It is a study of chunks of language which are bigger than a single sentence.

Dicourse language concerns inter-sentential links that is how the immediately preceding sentences affect the interpretation of the next sentence.

Discourse knowledge is important for interpreting pronouns and temporal aspects of the information conveyed.

7.WORLD KNOWLEDGE: World knowledge is nothing but everyday knowledge that all speakers share about the world. It includes the general knowledge about the structure of the world and what each language user must know about the other user's beliefs and goals. This essential to make the language understanding much better.

Q36. Explain in brief forward reasoning and backward reasoning . [9M]

Ans: Whether you use forward or backwards reasoning to sove a problem depends on the properties of your rule set and initial facts. Sometimes, if you have some particular goal (to test some hypothesis), then backward

chaining will be much more efficient, as you avoid drawing conclusions from irrelevant facts. However, sometimes backward chaining can be very wasteful - there may be many possible ways of trying to prove something, and you may have to try almost all of them before you find one that works. Forward chaining may be better if you have lots of things you want to prove (or if you just want to find out in general what new facts are true); when you have a small set of initial facts; and when there tend to be lots of different rules which allow you to draw the same conclusion. Backward chaining may be better if you are trying to prove a single fact, given a large set of initial facts, and where, if you used forward chaining, lots of rules would be eligible to fire in any cycle. A sequential type Fuzzy backward reasoning device is disclosed, which is capable of performing computation progressing every time a feature quantity is observed to update reasoning and of performing the reasoning even if the order of observations is arbitrary, by providing means for performing sequential reasoning instead of batch type reasoning means, and feedback means for feeding back a reasoned result.

Additionally, a target recognition device is disclosed which is capable of computing as a numerical value the reliable degree of a recognized result on a target by obtaining another recognizing information even if there is not obtained any information concerning the target, using said sequential type Fuzzy backward reasoning device.

Q37. Explain in brief forward chaining. **[6M]**

Ans:

Forward Chaining: An inference engine using forward chaining searches the inference rules until it finds one where the IF clause is known to be true. When found it can conclude, or infer, the THEN clause, resulting in the addition of new information to its dataset. In other words, it starts with some facts and applies rules to
find all possible conclusions. Therefore, it is also known as Data Driven Approach.

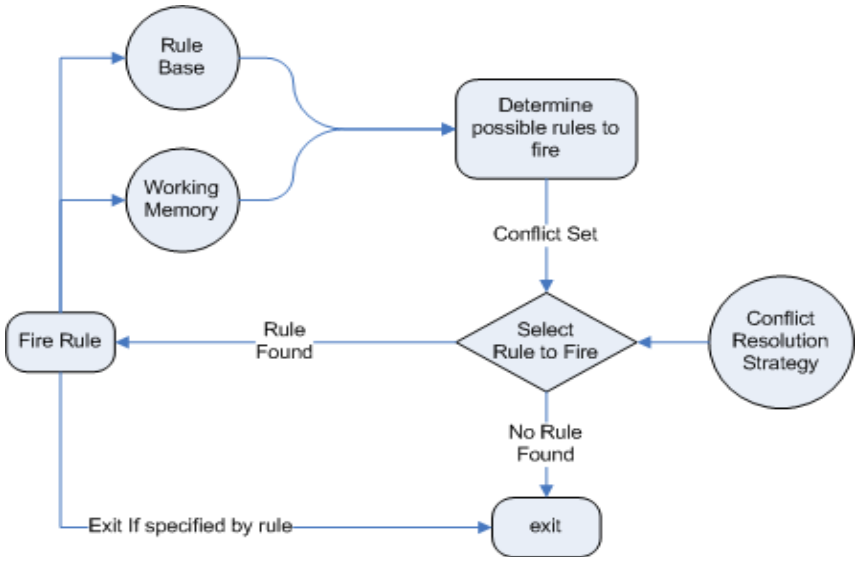


Fig: Forward chaining

Q38. What are different types of Knowledge Representation schemes. **[7M]**

Ans:- There are different types of knowledge representation schemes:-

(i) Logical Knowledge Representation Scheme:- Logical knowledge representation scheme includes expression.

e.g.;Predicate and propositional logic

(ii) Procedural Knowledge Representation Scheme:- Procedure Knowledge Representation Scheme includes procedures. E.g.:- Rule based system.

(iii) Network Knowledge Representation Scheme:- Network Knowledge Representation Scheme includes graphs.

e.g.:Semantic network.

(iv)Structured Knowledge Representation Scheme:- Structured Knowledge Representation Scheme includes graphs.

E.g:- Script, Frames etc.

Q39. Write short note on Procedural knowledge & Inheritable Knowledge. [7M]

Ans:-

Procedural Knowledge

Basic idea:

- Knowledge encoded in some procedures
  - small programs that know how to do specific things, how to proceed.
  - *e.g* a parser in a natural language understander has the knowledge that a *noun phrase* may contain articles, adjectives and nouns. It is represented by calls to routines that know how to process articles, adjectives and nouns.

Advantages:

- *Heuristic* or domain specific knowledge can be represented.
- *Extended logical inferences*, such as default reasoning facilitated.
- *Side effects* of actions may be modelled. Some rules may become false in time. Keeping track of this in large systems may be tricky.

Disadvantages:

- Completeness -- not all cases may be represented.
- Consistency -- not all deductions may be correct.

  e.g If we know that *Fred is a bird* we might deduce that *Fred can fly*. Later we might discover that *Fred is an emu*.

- Modularity is sacrificed. Changes in knowledge base might have far-reaching effects.
- Cumbersome control information.

Inheritable Knowledge:-

Simple relational knowledge :- The simplest way of storing facts is to use a relational method where each fact about a set of objects is set out systematically in columns. This representation gives little opportunity for inference, but it can be used as the knowledge basis for inference engines.

- Simple way to store facts.
- Each fact about a set of objects is set out systematically in columns.
- Little opportunity for inference.
- Knowledge basis for inference engines.

Musician	Style	Instrument	Age
Miles Davis	Jazz	Trumpet	deceased
John Zorn	Avant Garde	Saxophone	35
Frank Zappa	Rock	Guitar	deceased
John Mclaughlin	Jazz	Guitar	47

Figure: Simple Relational Knowledge

Inheritable knowledge

Relational knowledge is made up of objects consisting of

- attributes
- corresponding associated values.

We extend the base more by allowing inference mechanisms:

- Property inheritance
  - elements inherit values from being members of a class.
  - data must be organised into a hierarchy of classes

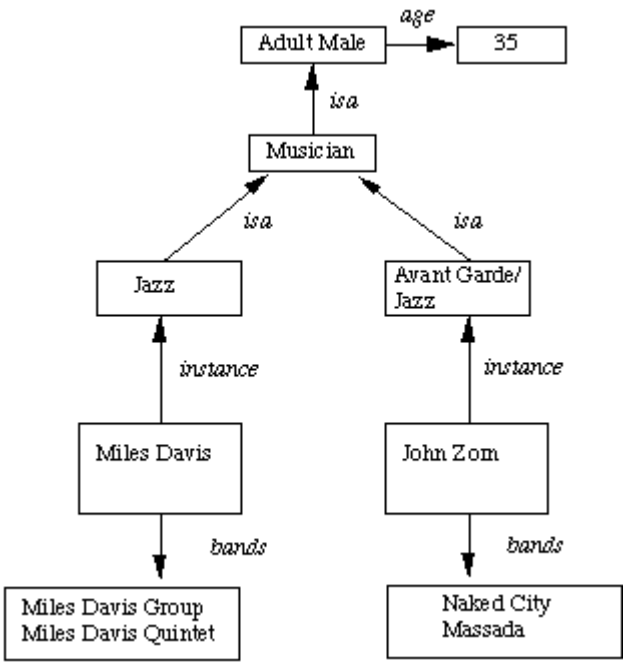


Figure:- Property Inheritance Hierarchy

- Boxed nodes -- objects and values of attributes of objects.

- Values can be objects with attributes and so on.
- Arrows -- point from object to its value.
- This structure is known as a slot and filler structure, semantic network or a collection of frames.

Q40. Explain means-ends analysis with example of Household Robot domain. Give rules and Difference Table for it. [7M]

Ans:-

Means-Ends Analysis process centers around the detection of differences between the current state and the goal state. Once such a difference is isolated, an operator that can reduce the difference must be found. If the operator cannot be applied to the current state, we set up a subproblem of getting to a state in which it can be applied.

The kind of backward chaining in which operators are selected and then subgoals are set up to establish the preconditions of the operators.

Operator	Preconditions	Results
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)	None	At(robot, loc)
PICKUP(Obj)	At(robot, obj)	Holding(obj)
PUTDOWN(obj)	Holding(obj)	¬holding(obj)
PLACE(Obj1, obj2)	At(robot, obj2)^ Holding(obj1)	On(obj1, obj2)

DIFFERENCE TABLE:-

	PUSH	Carry	Walk	Pickup	Putdown	Place
Move Obj	*	*				
Move robot			*			
Clear				*		

Obj						
Get object on obj						*
Get arm empty					*	*
Be holding obj				*		

Q41.

Ans: **Semantics Nets (Associative Network)**

A semantic network or a semantic net is a structure for representing knowledge as a pattern of interconnected nodes and arcs. It is also defined as a graphical representation of knowledge. The objects under consideration serve as nodes and the relationships with another nodes give the arcs.

In a semantic net, information is represented as a set of nodes connected to each other by a set of labeled ones, which represent relationships among the nodes. A fragment of a typical semantic net is shown in Figure

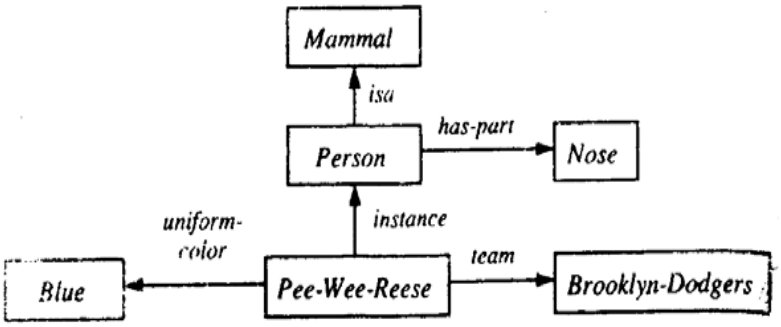


Figure: A Semantic Network

This network contains example of both the is a and instance relations, as well as some other, more domain-specific relations like team and uniform-color. In this network we would use inheritance to derive the additional relation.

has-part (Pee-Wee-Reese, Nose).

Frames

Marvin Minsky in the book on computer vision proposed frames as a means of representing common-sense knowledge. In that Minsky proposed that knowledge is organized into small “packets” called frames. The contents of the frame are certain slots, which have values. All frames of a given situation constitute the system. A frame can be defined as a data structure that has slots for various objects and a collection of frames consists of exceptions for a given situation. A frame structure provides facilities for describing objects, facts about situations, procedures on what to do when a situation is encountered. Because of this facilities a frames are used to represent the two types of knowledge, viz., declarative/factual and procedural.

In this example,
the frames Person, Adult-Male, ML-Baseball-Player (corresponding to major league baseball players) Pitter, and ML-Baseball-Team (for major league baseball team) are all classes. The frame Pee-Wee-Reese and Brooklyn-Dodgers are instances.
The is a relation that we have been using without a precise definition is in fact the subset relation. The isa of adult males is a subset of the set of people. The set of major league baseball players is a subset of the set of adult males, and so forth. Our instance relation corresponds to the relation element of Pee Wee Reese that is an element of the set of fielders. Thus he is also an element of all of the superset of fielders, including major league baseball players and people. The transitivity of isa that we have taken for granted in our description of property inheritance follows directly from the transitivity of the subset relation.
Both the isa and instance relations have inverse attributes, where we call subclasses and all-instances. We do not bother to write them explicitly in our examples unless we need to refer to them. We assume that the frame system maintains them automatically. Either explicitly or by computing them if necessary.