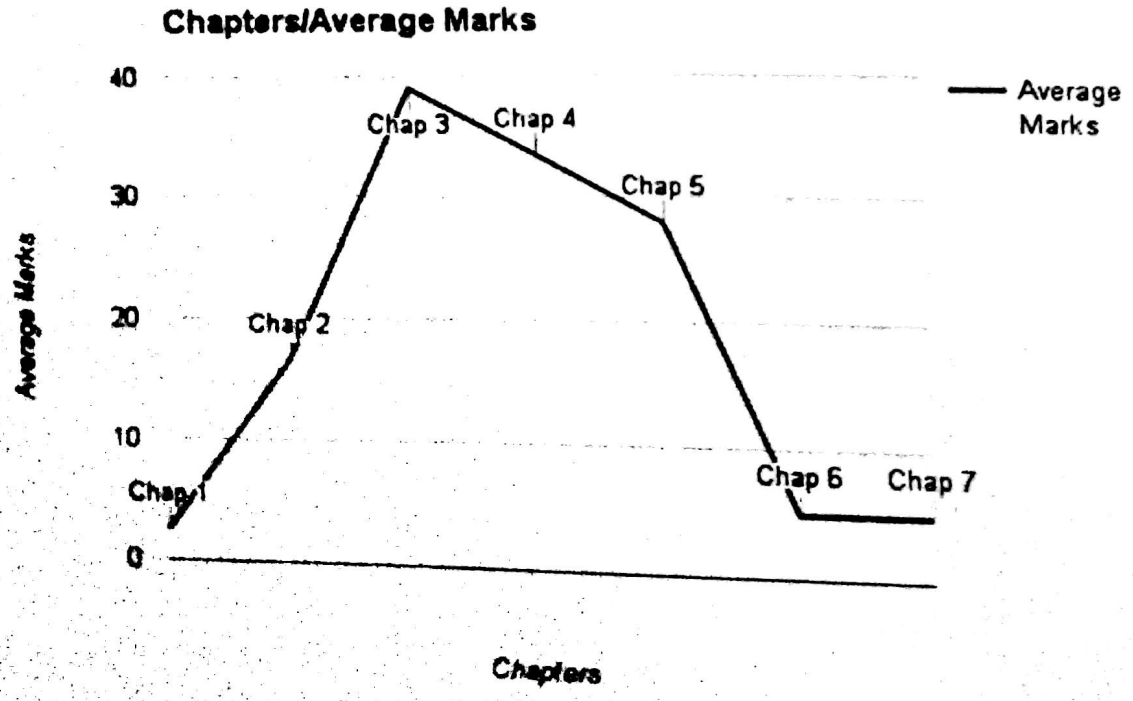
Distribution Matrix

Module	Dec 2015	May 2016
1. Introduction to Artificial Intelligence	-	5
2. Intelligent Agents	13	20
3. Problem solving	28	50
4. Knowledge and Reasoning	37	30
5. Planning and Learning	31	25
6. Applications	10	-
7. Miscellaneous	5	5

Overall Weightage



Chap 1| Introduction to Artificial Intelligence

Q1) Explain Turing test designed for satisfactory operational definition of intelligence.

(5M | May 2016)

Ans:

The Turing Test, proposed by Alan Turing (195O), was designed to provide a satisfactory operational definition of intelligence.

- Rather than proposing a long and perhaps controversial list of qualifications required for intelligence, he suggested a test based on in distinguishability from undeniably intelligent entities-human beings.
- The computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or not.
- For now, we note that programming a computer to pass the test provides plenty to work on.

The computer would need to possess the following **capabilities**:

- **Natural language processing** to enable it to communicate successfully in English.
- **Knowledge representation** to store what it knows or hears;
- **Automated reasoning** to use the stored information to answer questions and to draw new conclusions;
- **Machine learning** to adapt to new circumstances and to detect and extrapolate patterns.

Turing's test deliberately avoided direct physical interaction between the interrogator and the computer, because physical simulation of a person is unnecessary for intelligence.

However, the so-called **total Turing Test** includes a video signal so that the interrogator can test the

subject's perceptual abilities, as well as the opportunity for the interrogator to pass physical objects "through the hatch."

To pass the total Turing Test, the computer will need

- **Computer vision** to perceive objects, and
- **Robotics** to manipulate objects and move about.

Chap 2| Intelligent Agents

Q1) Define Rationality and Rational Agent. Give an example of rational action performed by any intelligent agent. **(5M | Dec 2015)**

Ans:

Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment.

Rational agent is the one which has the ability to perceive from the environment and act on it so that the performance measure can be maximized i.e., every entry in the table for the agent function is filled out correctly.

Example: Intelligent Agent – Simple Vacuum Cleaner
Rational Action – cleans a square if it is dirty and moves to the other square if not

Q2) Give the PEAS description for an Internet shopping agent. Characterize its environment. **(5M | Dec 2015, May 2016)**

Ans:

PEAS description

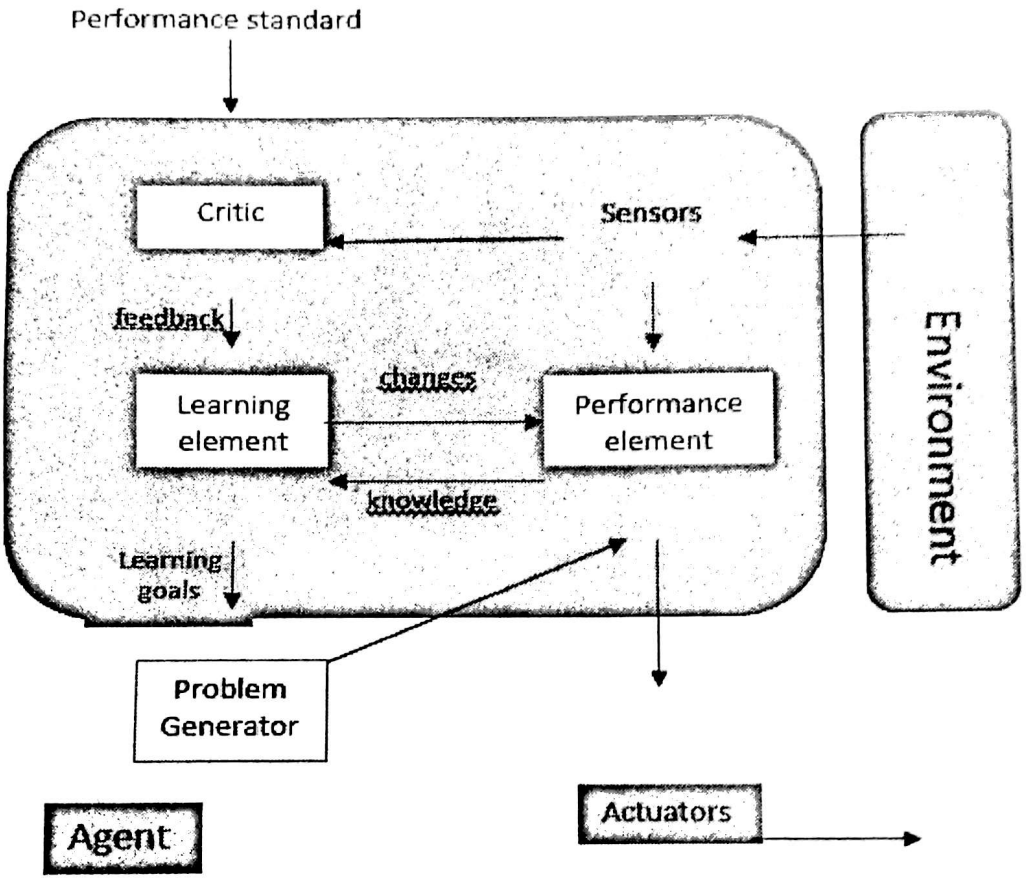
Performance measure: price, quality, appropriateness, efficiency
Environment: current and future WWW sites, vendors, shippers
Actuators: display to user, follow URL, fill in form
Sensors: HTML pages (text, graphics, scripts)

Environment Characteristics:

Fully observable: No
Deterministic: Partly // partially observable
Episodic: No // sequential
Static: Semi // the world changes partly while the agent is thinking
Discrete: Yes
Single-agent: No // multi-agent

Q3) What are the basic building blocks of Learning Agent? Explain each of them with a neat block diagram.

Ans: **(8M,10M | Dec 2015, May 2016)**



A learning agent can be divided into four conceptual components which are the **basic building blocks of Learning Agent**

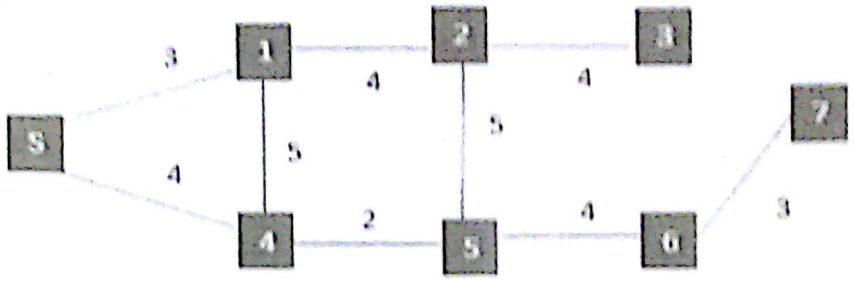
1. **Performance element** is responsible for selecting external actions. The performance element takes in percepts and decides on actions
2. **Learning element** is responsible for making improvements. The learning element uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future. The design of the learning element depends very much on the design of the performance element.
3. **Critic** tells the learning element how well the agent is doing with respect to a fixed performance standard. The critic is necessary because the percepts themselves provide no indication of the agent's success. For example, a chess program could receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; the percept itself does not say so. It is important that the performance

standard be fixed. Conceptually, one should think of it as being outside the agent altogether, because the agent must not modify it to fit its own behaviour

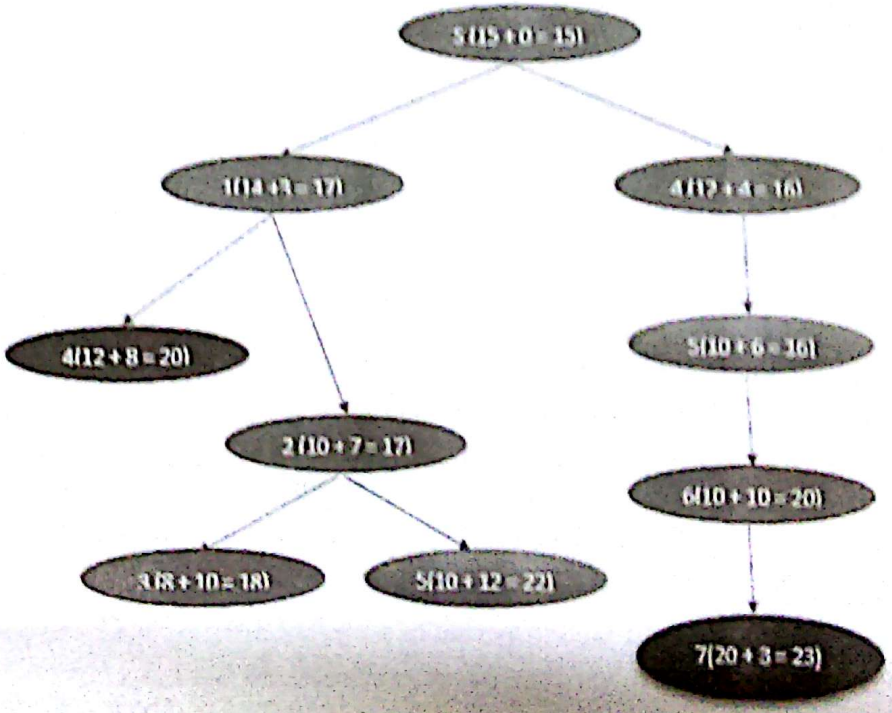
Problem generator is responsible for suggesting actions that will lead to new and informative experiences. The point is that if the performance element had its way, it would keep doing the actions that are best, given what it knows. But if the agent is willing to explore a little, and do some perhaps suboptimal actions in the short run, it might discover much better actions for the long run. The problem generator's job is to suggest these exploratory actions.

Chap 3| Problem Solving

Q1) Consider the graph given in Figure 1 below. Assume that the initial state is S and goal state is 7. Find a path from the initial state to the goal state using A* Search. Also report the solution cost. The straight line distance heuristic estimates for the nodes are as follows:
h(1) = 14, h(2) = 10, h(3) = 8, h(4) = 12, h(5) = 10, h(6) = 10, h(8) = 15.



Ans: (10M | Dec 2015)



Result: Path = S-4-5-6-7
Solution cost = 23

Q2) What are the problems/frustrations that occur in hill climbing technique? Illustrate with an example. **(6M | Dec 2015)**

Ans:

1. **Cannot recover from failure:** Hill-climbing strategies expand the current state of the search and evaluate its children. The best child is selected for further expansion; neither its siblings nor its parent are retained. Because it keeps no history, the algorithm cannot recover from failures of its strategy.

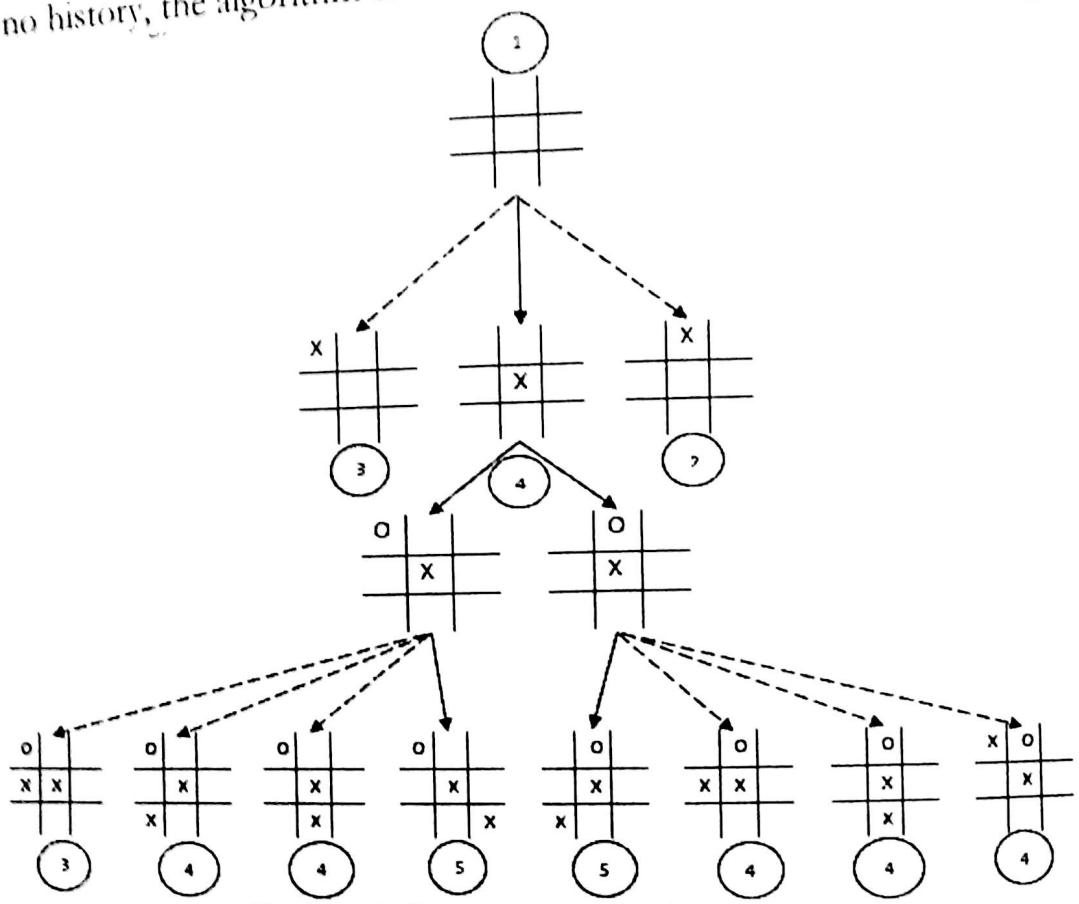


Fig. Heuristically reduced state space for Tic-Tac-Toe
(The 'Most Wins' heuristic applied to the first children.)

2. **Stuck at local Maxima:** Hill-climbing strategies have a tendency to become stuck at local maxima. If they reach a state that has a better evaluation than any of its children, the algorithm halts.
If this state is not a goal, but just a local maximum, the algorithm may fail to find the best solution.
That is, performance might well improve in a limited setting, but because of the shape of the entire space, it may never reach the overall best.
An example of local maxima in games occurs in the 8-puzzle. Often, in order to move a particular tile to its destination, other tiles already in goal position need be moved out. This is necessary to solve the puzzle but temporarily worsens the board state. Because "better" need not be "best" in an absolute

sense, search methods without backtracking or some other recovery mechanism are unable to distinguish between local and global maxima.

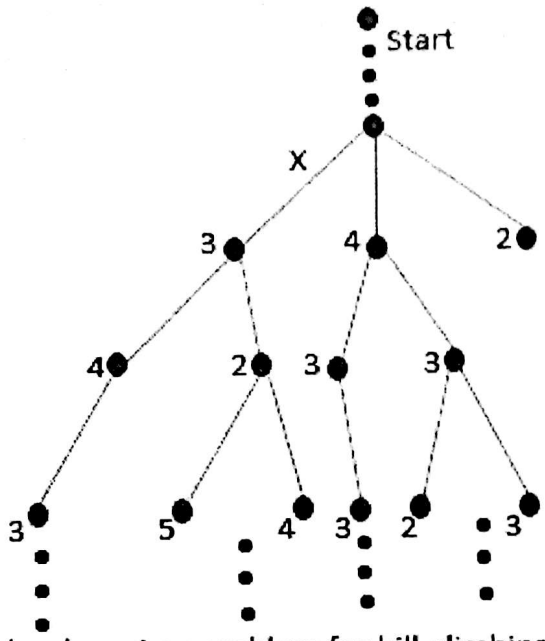


Fig. The local maxima problem for hill-climbing with 3-level look ahead

3. **Multiple Local Maxima:** hill-climbing functions can have multiple local maxima, which frustrates hill-climbing methods. For example, in 8-puzzle problem consider the goal state & initial state as below

1	2	3
	7	4
8	6	5

Goal State

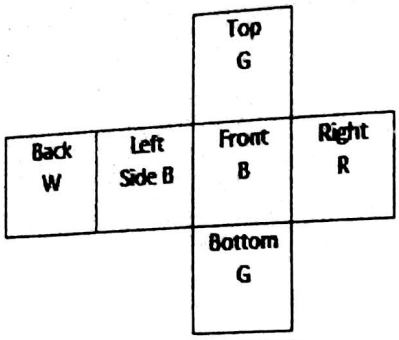
1	2	5
	7	4
8	6	3

Initial State

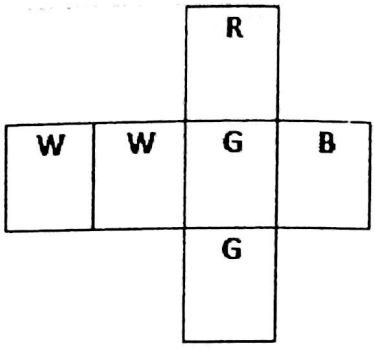
Any applicable rule applied to the initial state description lowers value of our hill-climbing function. In this case the initial state description is a local (but not a global) maximum of the function.

4. **Stuck on plateaus & ridges:** The hill climbing algorithm may get the problem stuck on plateaus & ridges

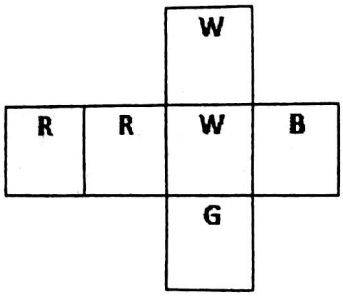
5. **End-bunching:** Instant insanity problem where the goal of the puzzle is to arrange the cubes one on top of the other in such a way that they form a stack four cubes high, with each of the four sides having exactly one red, one blue, one green, & one white cubes.



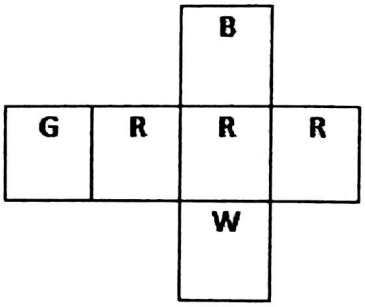
2B2G



2G2W



2R2W



3R

6. The goal state could be characterized exactly as having each of the four colours represented on each of the four vertical sides. Hence, we may consider the beginning state to have the evaluation vector (4, 4, 4, 4). The four dimensions could rather naturally be combined into a one-dimensional evaluation function simply by summing the four components. In obtaining this sum, it seems natural to give equal weight to each component, since each dimension has the same range of values and an analogous meaning. Hill climbing here greatly reduces the search space, but the method still leaves a very large number of alternatives to investigate. There are many equivalent options at each of the four nonterminal nodes of the state-action tree for Instant Insanity, so hill climbing with this evaluation function hardly yields the answer with a single series of four choices. The difficulty with this state evaluation function applied to this problem is that it is much harder to increase the evaluation function by the required amount at the last (fourth) choice node than at earlier nodes. At most of the last nodes, no action will achieve the goal, even though the solver is currently at a node that has the evaluation (3, 3, 3, 3). Whether or not you can solve the problem is determined by existence of such an action at the fourth node, but the

evaluation function for the states that could be achieved at earlier nodes gives very inadequate information concerning the "correct" fourth node at which to be. That is, there are many fourth nodes with the evaluation (3, 3, 3, 3), and very few of these have any action that leads to a terminal node with the evaluation (4, 4, 4, 4).

There are many problems like this, where the restrictions bunch up at the end of the problem. It is as if you had many easy trails to climb most of the way up a mountain, but the summit was attainable from only a few of these trails, with the rest running into unscalable precipices. Hill climbing is often not a very good method to use in such cases, though it may considerably reduce the amount of trail-and-error search.

The end-bunching of restrictions is a difficulty with hill climbing that is somewhat analogous to the local maximum difficulty.

7. **Detours and Circling:** Problems with multiple equivalently valued paths at the early nodes can be difficult to solve with hill climbing, but perhaps the greatest frustration in using the method comes in detour problems, where at some node you must actually choose an action that decreases the evaluation. Somewhat less difficulty is encountered in what might be called circling problems, where at one or more nodes you must take actions that do not increase the evaluations. If the nodes where you must detour or circle have no better choices (that is, no choices that increase the evaluation), then you are more likely to try detouring or circling than if the critical nodes have better choices. When better choices are available, you tend to just choose them and go on without considering the possibility of detouring or circling. If the path you choose does not lead to the goal, you might go back and investigate alternative paths, but the first ones to be investigated will be those that were equivalent or almost equivalent at some previous node. Only after all of this fails should you try detouring – that is, choosing an action at some node that produces a state that has a lower evaluation than the previous state had.

   The missionaries-and-cannibals problem is a famous example of the difficulties encountered by hill climbing in detour problem.

8. **Inference Problems:** The description of the problem state must generally be considered to include the entire set of expressions given or derived up to that point. Since the goal isusually asingleexpression, it is generally much more difficult todefine an evaluation function that isuseful for hill climbing that compares the current state with the goalstate. Another reason for the

greater difficulty in using hill climbing in inference problems is that the non-destructive operations frequently found in such problems are often not one-to-one operations-that is, operations that take one expression as input and produce one expression as output. There are such one-to-one operations, of course. However, in addition, inference problems usually contain a variety of two-to-one, and three-to-one, or even more complex operations-that is, operations that take two or three or more expressions as input and produce one expression as the output (the inferred expression).

Q3) Draw a game tree for a Tic-Tac-Toe problem. **(4M | Dec 2015)**

Ans:

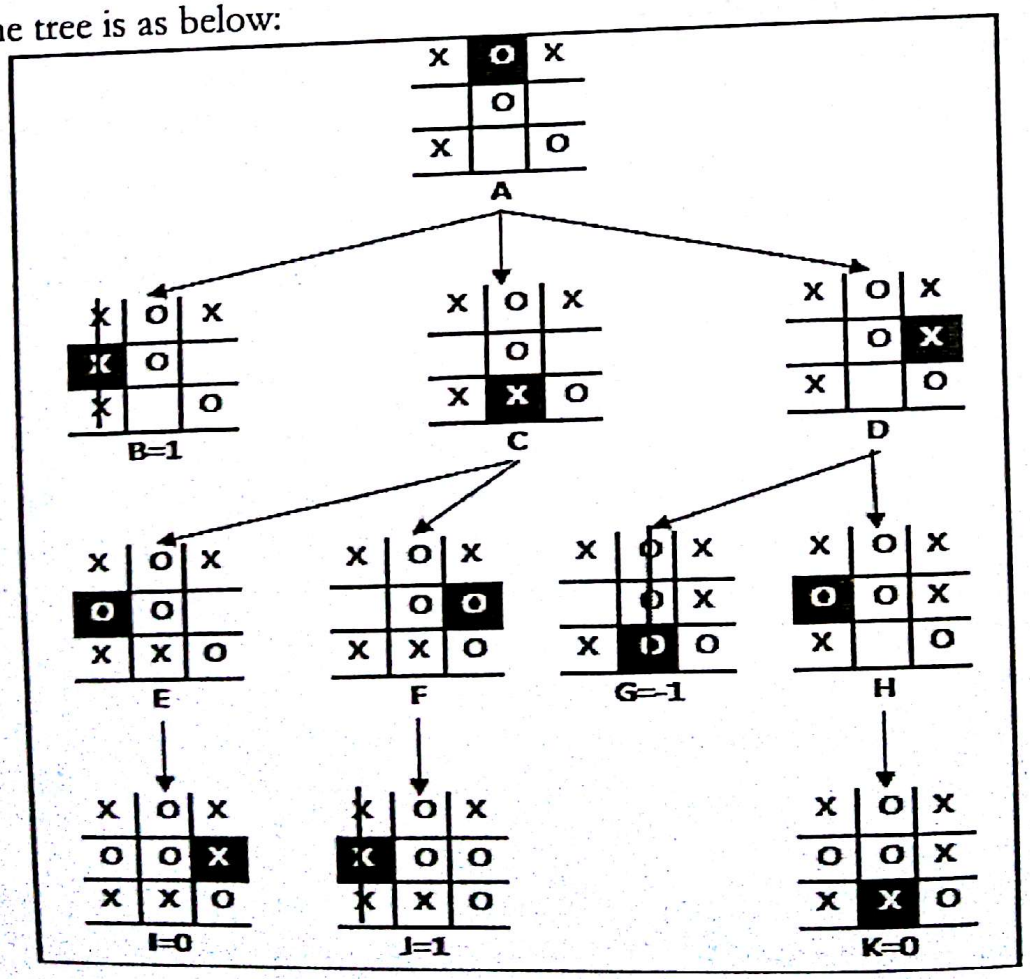
Consider A as intermediate state for initiating game tree where,

X is to play

Assume Goal State = 'X' Wins = 1

Thus, 'o' wins = -1 & draw = 0

The game tree is as below:



Q4) Write a short note on genetic algorithm.

(8M | Dec 2015)

Ans:

Genetic algorithms provide a natural model for parallelism because each neuron or segment of a solution is an independent unit.

- Genetic algorithms begin with a population of candidate problem solutions.
- Candidate solutions are evaluated according to their ability to solve problem instances: only the fittest survive and combine with each other to produce the next generation of possible solutions
- In the genetic algorithm model, for example, a population of patterns represents the candidate solutions to a problem.
- As the algorithm cycles, this population of patterns "evolves" through operations which mimic reproduction, mutation, and natural selection.
- Genetic algorithms are also applied to more complex representations, including production rules, to evolve rule sets adapted to interacting with an environment.
  For example, genetic programming combines and mutates fragments of computer code in an attempt.
- Like neural networks, genetic algorithms are based on a biological metaphor: they view learning as a competition among a population of evolving candidate problem solutions.
- A "fitness" function evaluates each solution to decide whether it will contribute to the next generation of solutions.
- Then, through operations analogous to gene transfer in sexual reproduction, the algorithm creates a new population of candidate solutions.

In the genetic algorithm, process is as follows:

Step 1. Determine the number of chromosomes, generation, and mutation rate and crossover rate value

Step 2. Generate chromosome-chromosome number of the population, and the initialization value of the genes chromosome-chromosome with a random value

Step 3. Process steps 4-7 until the number of generations is met

Step 4. Evaluation of fitness value of chromosomes by calculating objective function

Step 5. Chromosomes selection

Step 5. Crossover

Step 6. Mutation

Step 7. New Chromosomes (Offspring)

Step 8. Solution (Best Chromosomes)

Let P(t) define a population of candidate solutions, xi, at time t:

P(t) ~ {xi. xl, ..., x:J

We now present a **general form of the genetic algorithm:**

procedure genetic algorithm;

begin

set time t:~ 0;

initialize the population P(t);

while the termination condition is not met do

begin

evaluate fitness at each member at the population P(t);

select members from population P(t) based on fitness;

produce the offspring at these pairs using genetic operators;

replace, based on fitness, candidates of P(t), with these offspring;

set time t := t +1

end

end.

The flowchart of algorithm can be seen in figure below

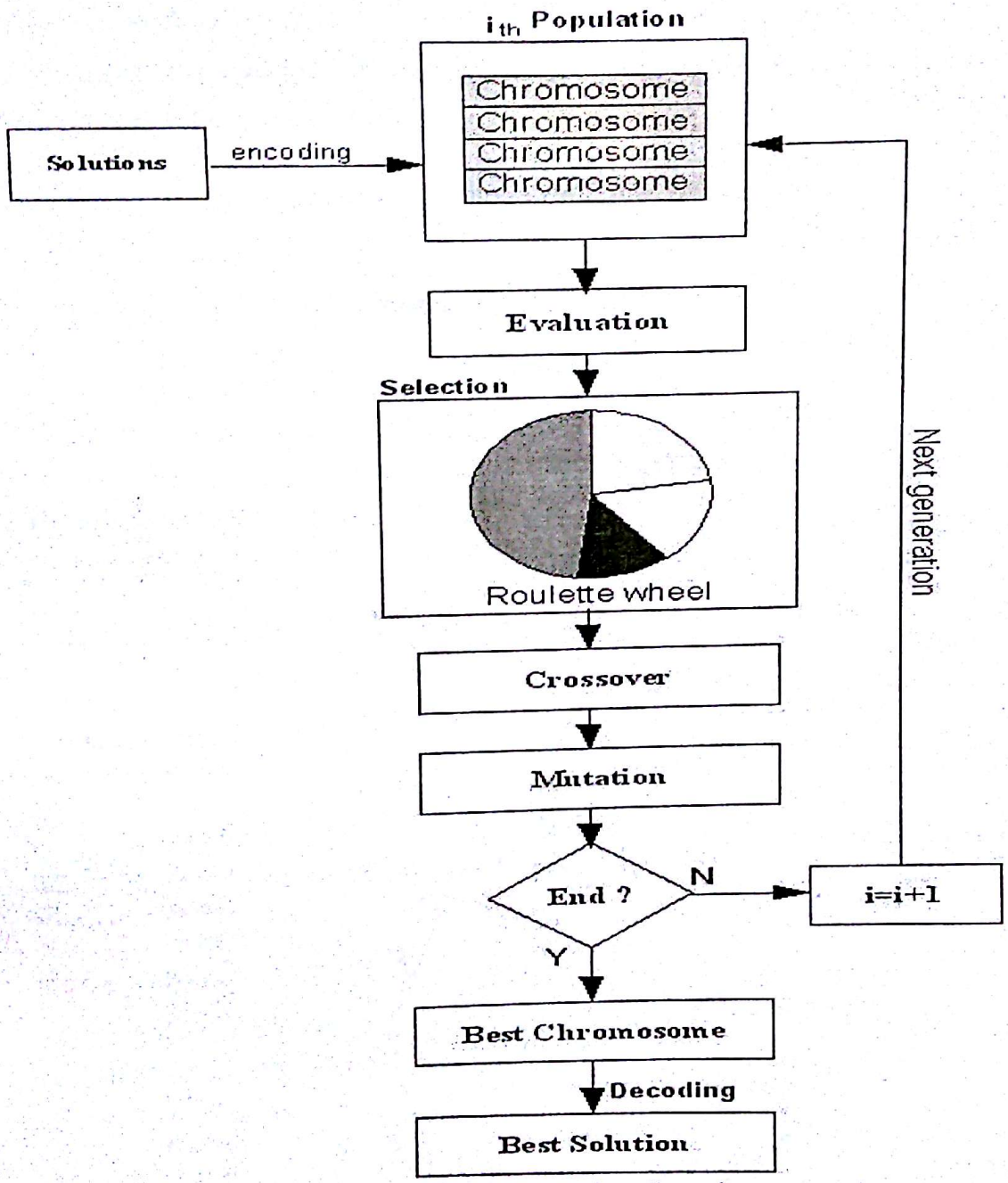


Fig. Genetic algorithm flowchart

This algorithm articulates the basic framework of genetic learning; specific implementations
of the algorithm instantiate that framework in different ways.

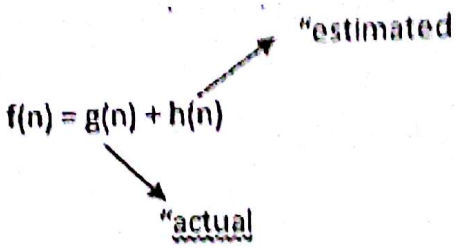
- What percentage of the population is retained?
- What percentage mate and produce offspring?
- How often and to whom are the genetic operators applied?

Q5) Explain Hill-climbing algorithm with an example

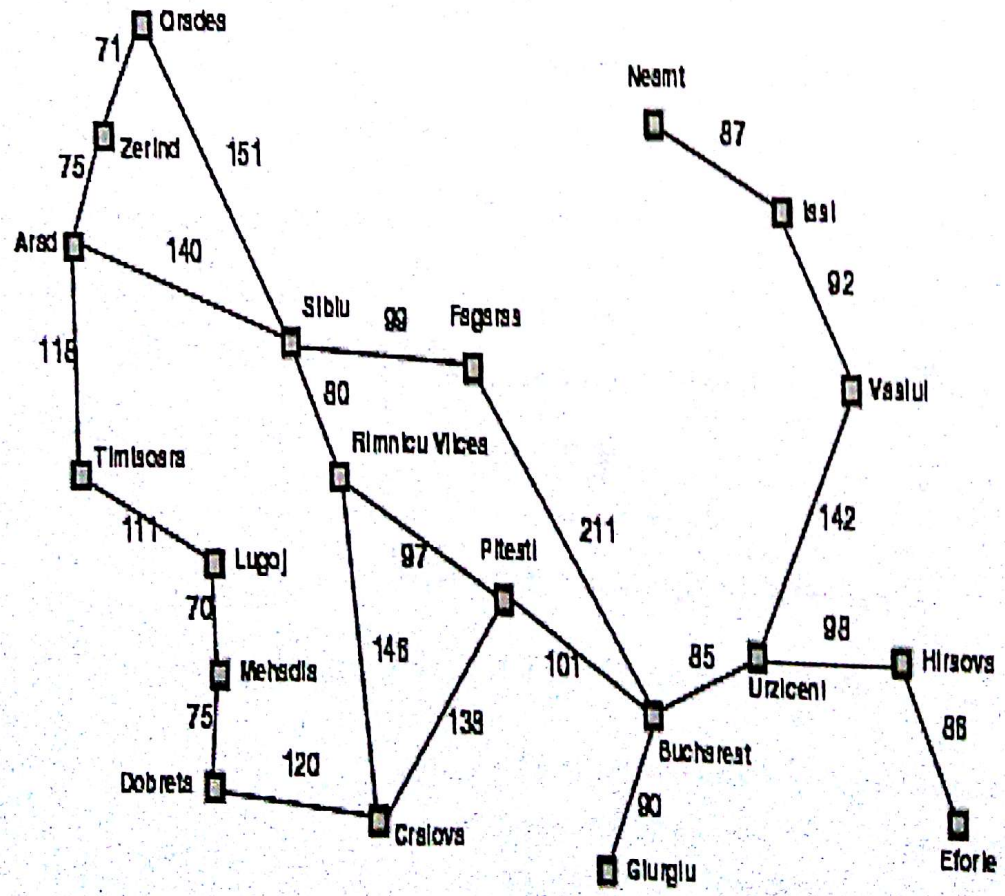
(5M | May 2016)

Ans:

- **Difficulty:** we want to still be able to generate the path with minimum cost
- A* is an algorithm that:
  - Uses heuristic to guide search
  - While ensuring that it will compute a path with minimum cost
- A* computes the function



- f(n) = g(n) + h(n)
  - g(n) = "cost from the starting node to reach n"
  - h(n) = "estimate of the cost of the cheapest path from n to the goal node"
- Example: minimize f(n) = g(n) + h(n)



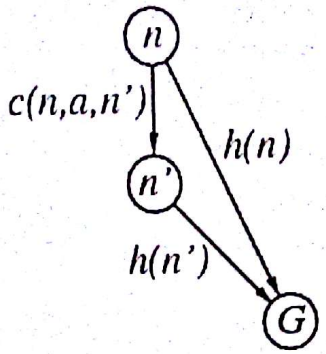
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Q6) Prove that A* is admissible if it uses a monotone heuristic.

Ans: **(5M | May 2016)**

A heuristic is **consistent or monotone** if for every node n, every successor n' of n generated by any action a,

$h(n) \leq c(n,a,n') + h(n')$



Theorem:

If h(n) is consistent, A* using GRAPH-SEARCH (keeps all checked nodes in memory to avoid repeated states) is optimal

Proof: If h is consistent, we have

$f(n') = g(n') + h(n')$ (by def.)

$= g(n) + c(n,a,n') + h(n')$ $(g(n')=g(n)+c(n.a.n'))$

$\geq g(n) + h(n)$

$= f(n)$ (consistency)

$f(n') \geq f(n)$

i.e., f(n) is non-decreasing along any path.

Q7) Compare following informed searching algorithms based on performance measure with justification: Complete, Optimal, Time complexity and space complexity.

a. **Greedy best first**
b. **A***
c. **Recursive best-first (RBFS)**

Ans: **(10M | May 2016)**

Parameters	Greedy Best First	A*	Recursive best-first (RBFS)
Complete	No – can get stuck in loops, e.g., Iasi	Yes (unless there are infinitely many nodes	Yes, similar to A*.

	Neamt Iasi Neamt	with f ≤ $f(G)$)	
Optimal	No	Yes (depending upon search algo and heuristic property)	Yes, similar to A*.
Time Complexity	*O(bm)*, but a good heuristic can give dramatic improvement	Exponential	The time complexity is difficult to characterize: it depends both on the accuracy of the heuristic function and on how often the best path changes as nodes are expanded. Each mind change corresponds to an iteration of IDA_, and could require many re-expansions of forgotten nodes to recreate the best path and extend it one more node. RBFS is somewhat more efficient than IDA_, but still suffers from excessive node regeneration.
Space Complexity	*O(bm)* --keeps all nodes in memory	Keeps all nodes in memory	O(bd) (linear space best-first search)

Q8) Apply alpha-Beta pruning on example given in Figure 2 considering first node as max.

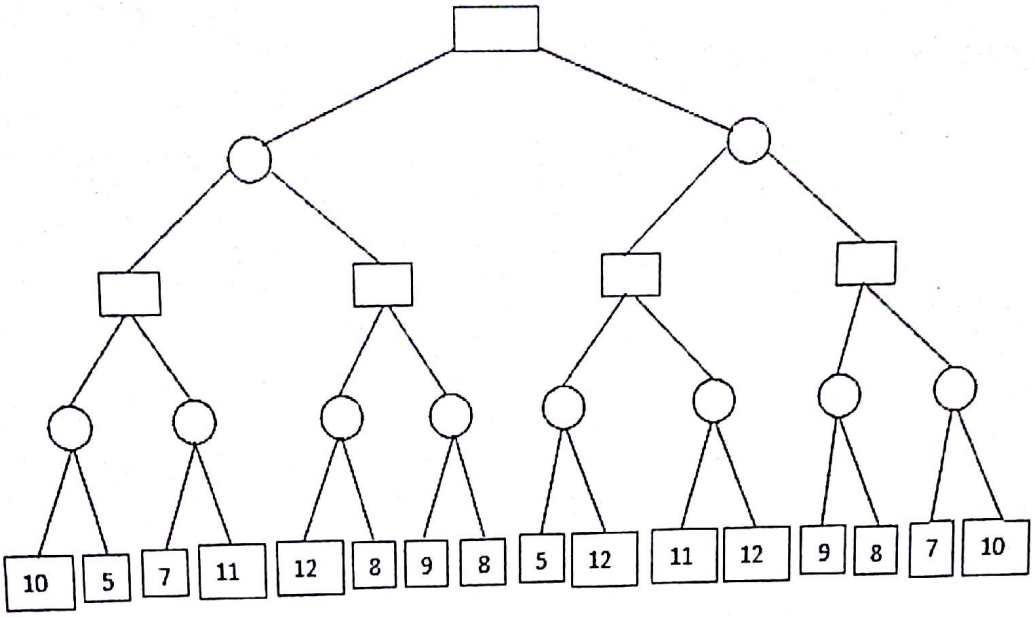


Figure 2

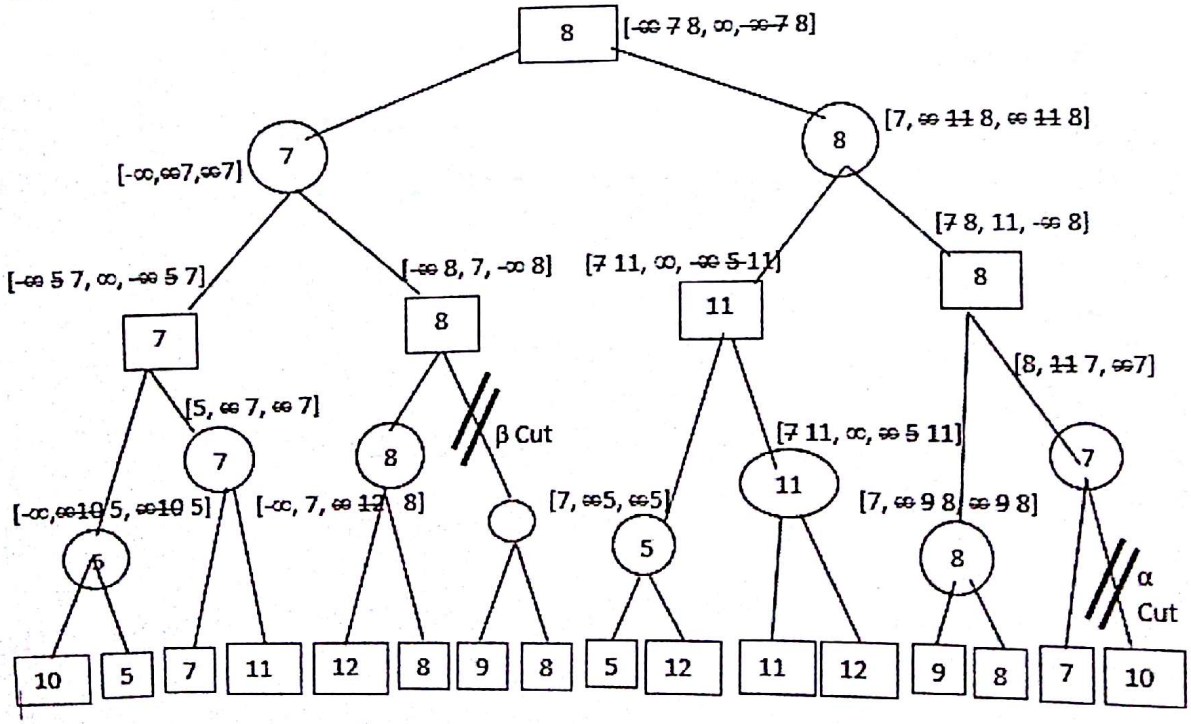
(10M | May 2016)

Ans:

Consider [α, β, V] at each node

For MAX Nodes, V=max(α, β) & α >= -∞

For MIN Nodes, V=min(α, β) & β >= ∞



Q9) Explain how genetic algorithm can be used to solve a problem by taking a suitable example.

Ans: **(10M | May 2016)**

Problem Solving using genetic algorithm

Example: Suppose there is equality a +2 b +3 c +4 d = 30, genetic algorithm will be used to find the value of a, b, c, and d that satisfy the above equation.

Objective: Minimize the value of function f(x) where f (x) = ((a + 2b + 3c + 4d) - 30). Since there are four variables in the equation, namely a, b, c, and d, we can compose the chromosome as follow:

A	B	C	D

To speed up the computation, we can restrict that the values of variables a, b, c, and d are integers between 0 and 30.

Step 1. Initialization

For example we define the number of chromosomes in population are 6, then we generate random value of gene a, b, c, d for 6 chromosomes

Chromosome[1] = [a;b;c;d] = [12;05;23;08]
Chromosome[2] = [a;b;c;d] = [02;21;18;03]
Chromosome[3] = [a;b;c;d] = [10;04;13;14]
Chromosome[4] = [a;b;c;d] = [20;01;10;06]
Chromosome[5] = [a;b;c;d] = [01;04;13;19]
Chromosome[6] = [a;b;c;d] = [20;05;17;01]

Step 2. Evaluation

We compute the objective function value for each chromosome produced in initialization step:

F_obj[1] = Abs((12 + 2*05 + 3*23 + 4*08) - 30)=Abs((12 + 10 + 69 + 32) - 30)=Abs(123 - 30)= 93

F_obj[2] = Abs((02 + 2*21 + 3*18 + 4*03) - 30)=Abs((02 + 42 + 54 + 12) - 30)= Abs(110 - 30)= 80

F_obj[3] = Abs((10 + 2*04 + 3*13 + 4*14) - 30) = Abs((10 + 08 + 39 + 56) - 30)= Abs(113 - 30)= 83

F_obj[4] = Abs((20 + 2*01 + 3*10 + 4*06) - 30)= Abs((20 + 02 + 30 + 24) - 30)= Abs(76 - 30)= 46

F_obj[5] = Abs((01 + 2*04 + 3*13 + 4*19) - 30) = Abs((01 + 08 + 39 + 76) - 30)= Abs(124 - 30)= 94

F_obj[6] = Abs((20 + 2*05 + 3*17 + 4*01) - 30)= Abs((20 + 10 + 51 + 04) - 30)= Abs(85 - 30)= 55

Step 3. Selection

1. The fittest chromosomes have higher probability to be selected for the next generation. To compute fitness probability we must compute the fitness of each chromosome. To avoid divide by zero problem, the value of F_obj is added by 1.

Fitness[1] = 1 / (1+**F_obj**[1]) = 1 / 94 = 0.0106
Fitness[2] = 1 / (1+**F_obj**[2]) = 1 / 81 = 0.0123
Fitness[3] = 1 / (1+**F_obj**[3]) = 1 / 84 = 0.0119
Fitness[4] = 1 / (1+**F_obj**[4]) = 1 / 47 = 0.0213
Fitness[5] = 1 / (1+**F_obj**[5]) = 1 / 95 = 0.0105
Fitness[6] = 1 / (1+**F_obj**[6]) = 1 / 56 = 0.0179
Total = 0.0106 + 0.0123 + 0.0119 + 0.0213 + 0.0105 + 0.0179 = 0.0845

The probability for each chromosomes is formulated by: **P**[i] = **Fitness**[i] / **Total**

P[1] = 0.0106 / 0.0845 = 0.1254
P[2] = 0.0123 / 0.0845 = 0.1456
P[3] = 0.0119 / 0.0845 = 0.1408
P[4] = 0.0213 / 0.0845 = 0.2521
P[5] = 0.0105 / 0.0845 = 0.1243
P[6] = 0.0179 / 0.0845 = 0.2118

From the probabilities above we can see that Chromosome 4 that has the highest fitness, this chromosome has highest probability to be selected for next generation chromosomes. For the selection process we use roulette wheel, for that we should compute the cumulative probability values:

C[1] = 0.1254
C[2] = 0.1254 + 0.1456 = 0.2710
C[3] = 0.1254 + 0.1456 + 0.1408 = 0.4118
C[4] = 0.1254 + 0.1456 + 0.1408 + 0.2521 = 0.6639
C[5] = 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243 = 0.7882
C[6] = 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243 + 0.2118 = 1.0

Having calculated the cumulative probability of selection process using roulette-wheel can be done. The process is to generate random number R in the range 0-1 as follows.

R[1] = 0.201
R[2] = 0.284
R[3] = 0.099
R[4] = 0.822
R[5] = 0.398
R[6] = 0.501

If random number R [1] is greater than P [1] and smaller than P [2] then select Chromosome [2] as a chromosome in the new population for next generation:

NewChromosome[1] = **Chromosome**[2]
NewChromosome[2] = **Chromosome**[3]
NewChromosome[3] = **Chromosome**[1]
NewChromosome[4] = **Chromosome**[6]
NewChromosome[5] = **Chromosome**[3]
NewChromosome[6] = **Chromosome**[4]

Chromosome in the population thus became:

Chromosome[1] = [02;21;18;03]
Chromosome[2] = [10;04;13;14]
Chromosome[3] = [12;05;23;08]
Chromosome[4] = [20;05;17;01]
Chromosome[5] = [10;04;13;14]
Chromosome[6] = [20;01;10;06]

Step 4. Crossover

In this example, we use one-cut point, i.e. randomly select a position in the parent chromosome then exchanging sub-chromosome. Parent chromosome which will mate is randomly selected and the number of mate Chromosomes is controlled using crossover_rate (ϱc) parameters. Pseudo-code for the crossover process is as follows:

```
begin
k← 0;
while(k<population) do
R[k] ← random(0-1);
if (R[k] < ϱc ) then
select Chromosome[k] as parent;
end;
k = k + 1;
end;
end;
```

Chromosome k will be selected as a parent if R [k] <ϱc. Suppose we set that the crossover rate is 25%, then Chromosome number k will be selected for crossover if random generated value for Chromosome k below 0.25. The process is as follows: First we generate a random number R as the number of population.

R[1] = 0.191
R[2] = 0.259
R[3] = 0.760
R[4] = 0.006
R[5] = 0.159
R[6] = 0.340

For random number R above, parents are Chromosome [1], Chromosome [4] and Chromosome [5] will be selected for crossover.

Chromosome[1] >< **Chromosome**[4]
Chromosome[4] >< **Chromosome**[5]
Chromosome[5] >< **Chromosome**[1]

After chromosome selection, the next process is determining the position of the crossover point. This is done by generating random numbers between 1 to (length of Chromosome – 1). In this case, generated random numbers should be between 1 and 3. After we get the crossover point, parents Chromosome will be cut at crossover point and its gens will be interchanged. For example we generated 3 random number and we get:

C[1] = 1
C[2] = 1
C[3] = 2

Then for first crossover, second crossover and third crossover, parent's gens will be cut at gen number 1, gen number 1 and gen number 3 respectively, e.g.

Chromosome[1] = **Chromosome**[1] >< **Chromosome**[4] = [02;21;18;03] >< [20;05;17;01]
= [02;05;17;01]
Chromosome[4] = **Chromosome**[4] >< **Chromosome**[5] = [20;05;17;01] >< [10;04;13;14]
= [20;04;13;14]
Chromosome[5] = **Chromosome**[5] >< **Chromosome**[1] = [10;04;13;14] >< [02;21;18;03]
= [10;04;18;03]

Thus Chromosome population after experiencing a crossover process:

Chromosome[1] = [02;05;17;01]
Chromosome[2] = [10;04;13;14]
Chromosome[3] = [12;05;23;08]
Chromosome[4] = [20;04;13;14]
Chromosome[5] = [10;04;18;03]
Chromosome[6] = [20;01;10;06]

Step 5. Mutation

Number of chromosomes that have mutations in a population is determined by the mutation rate parameter. Mutation process is done by replacing the gen at random position with a new value. The process is as follows. First we must calculate the total length of gen in the population. In this case the total length of gen is total_gen = number_of_gen_in_Chromosome * number of population = 4 * 6 = 24

Mutation process is done by generating a random integer between 1 and total_gen (1 to 24). If generated random number is smaller than

mutation_rate(ρm) variable then marked the position of gen in chromosomes. Suppose we define ρm 10%, it is expected that 10% (0.1) of total_gen in the population that will be mutated:

number of mutations = 0.1 * 24 = 2.4 ≈ 2

Suppose generation of random number yield 12 and 18 then the chromosome which have mutation are Chromosome number 3 gen number 4 and Chromosome 5 gen number 2. The value of mutated gens at mutation point is replaced by random number between 0-30. Suppose generated random number are 2 and 5 then Chromosome composition after mutation are:

Chromosome[1] = [02;05;17;01]
Chromosome[2] = [10;04;13;14]
Chromosome[3] = [12;05;23;**02**]
Chromosome[4] = [20;04;13;14]
Chromosome[5] = [10;**05**;18;03]
Chromosome[6] = [20;01;10;06]

Finishing mutation process then we have one iteration or one generation of the genetic algorithm. We can now evaluate the objective function after one generation:

Chromosome[1] = [02;05;17;01]
F_obj[1] = Abs((02 + 2*05 + 3*17 + 4*01) - 30) = Abs((2 + 10 + 51 + 4) - 30) = Abs(67 - 30) = 37

Chromosome[2] = [10;04;13;14]
F_obj[2] = Abs((10 + 2*04 + 3*13 + 4*14) - 30) =Abs((10 + 8 + 33 + 56) - 30) =Abs(107 - 30)= 77

Chromosome[3] = [12;05;23;02]
F_obj[3] = Abs((12 + 2*05 + 3*23 + 4*02) - 30)= Abs((12 + 10 + 69 + 8) - 30)= Abs(87 - 30) = 47

Chromosome[4] = [20;04;13;14]
F_obj[4] = Abs((20 + 2*04 + 3*13 + 4*14) - 30)= Abs((20 + 8 + 39 + 56) - 30)= Abs(123 - 30)= 93

Chromosome[5] = [10;05;18;03]
F_obj[5] = Abs((10 + 2*05 + 3*18 + 4*03) - 30)= Abs((10 + 10 + 54 + 12) - 30)= Abs(86 - 30)= 56

Chromosome[6] = [20;01;10;06]

F_obj[6] = Abs((20 + 2*01 + 3*10 + 4*06) - 30)= Abs((20 + 2 + 30 + 24) - 30)= Abs(76 - 30)= 46

From the evaluation of new Chromosome we can see that the objective function is decreasing, this means that we have better Chromosome or solution compared with previous Chromosome generation. New Chromosomes for next iteration are:

Chromosome[1] = [02;05;17;01]
Chromosome[2] = [10;04;13;14]
Chromosome[3] = [12;05;23;02]
Chromosome[4] = [20;04;13;14]
Chromosome[5] = [10;05;18;03]
Chromosome[6] = [20;01;10;06]

These new Chromosomes will undergo the same process as the previous generation of Chromosomes such as evaluation, selection, crossover and mutation and at the end it produce new generation of Chromosome for the next iteration. This process will be repeated until a predetermined number of generations. For this example, after running 50 generations, best chromosome is obtained:

Chromosome = [07; 05; 03; 01]

This means that:

a = 7, b = 5, c = 3, d = 1

If we use the number in the problem equation

a +2 b +3 c +4 d = 30

7 + (2 * 5) + (3 * 3) + (4 * 1) = 30

We can see that the value of variable a, b, c and d generated by genetic algorithm can satisfy that equality.

Q10) Consider the graph given in Figure 3 below. Assume that the initial stage A and the goal state is G. Find a path from the initial state to the goal state using DFS. Also report the solution cost.

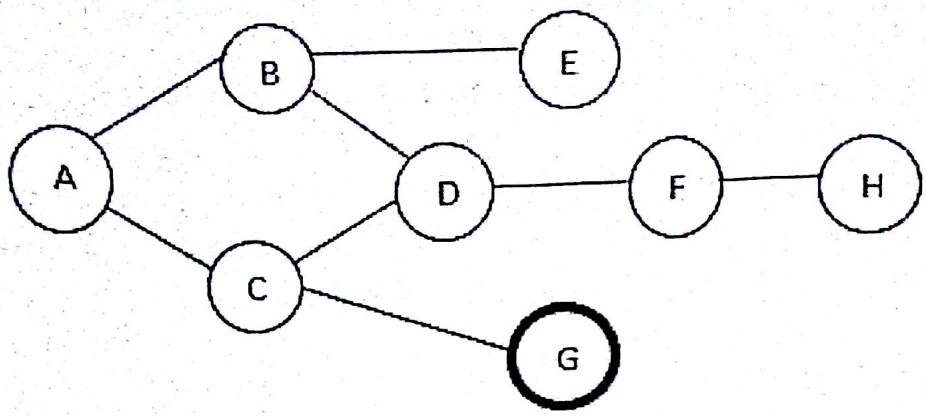
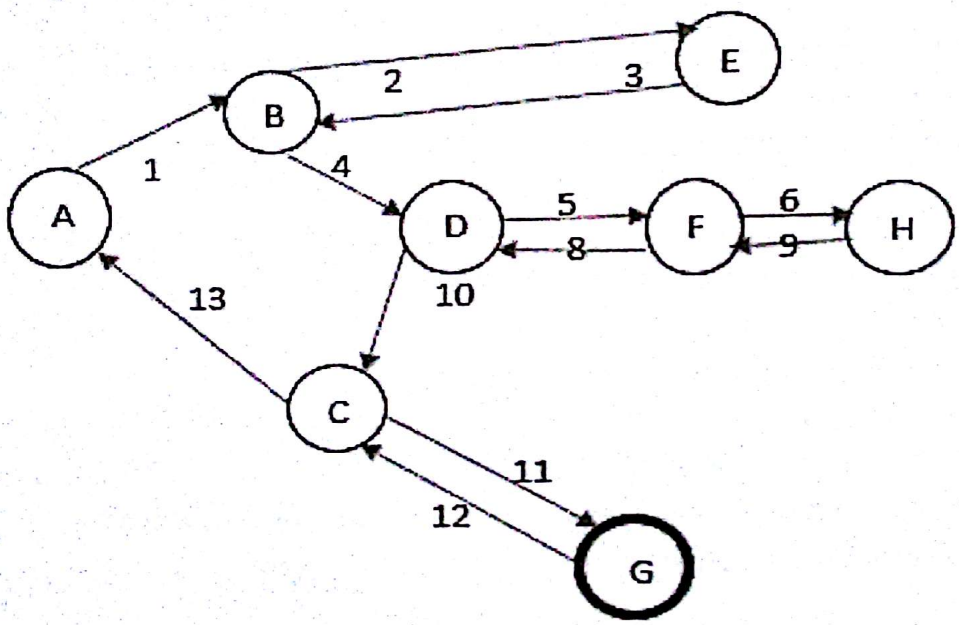


Figure 3.

Ans: **(10M | May 2016)**



Path: [A-B-E-B-D-F-H-F-D-C-G-C-A]

Solution path : A-C-G , Cost = '2'(AC + CG)

Chap 4| Knowledge and Reasoning

Q1) Represent the following statement into FOPL.

(i)Anyone who kills an animal is loved by no one.

(ii)A square is breezy if and only if there is a pit in a neighbouring square:

(Assume the wumpus world environment).

Ans: (5M | Dec 2015)

(i) Anyone who kills an animal is loved by no one.

Ans: Object constant: Person x

Predicates: Animal (y), Kills (x, y), loves (z, x)

FOPL: $\forall x ((\exists y \text{ Animal } (y) \wedge \text{ Kills } (x, y)) \Rightarrow \forall z \neg \text{ loves } (z, x))$

(ii)A square is breezy if and only if there is a pit in a neighbouring square

(Assume the wumpus world environment)

Ans: Object constant: Square s = [x, y]

Predicates: Pit(s), Breezy(s), Adjacent(s, r)

FOPL: $\forall s$ Breezy(s) $\Leftrightarrow \exists r$ (Adjacent(r, s) $\wedge$ Pit(r))

Q2) Convert the followed propositional logic statement into CNF

A-> (B<->C)

(4M|Dec 2015)

Ans:

A -> (B<->C) //KB

= ~A | (B<->C) //Eliminating Implication

= ~A | [(~B | C) & (B | ~C)] // Eliminating bidirectional Implication

= [~A | (~B | C)] & [~A | (B | ~C)] // Distributing & over |

= (~A | ~B | C) & (~A | B | ~C) // Flattening nested conjuncts & disjuncts

≈ $(\neg A \vee \neg B \vee C) \wedge (\neg A \vee B \vee \neg C)$

Q3) Consider the following axioms:

All people who are graduating are happy.

All happy people smile.

Someone is graduating.

Explain the following:-

1) Represent these axioms in first order predicate logic.

2) Convert each formula to clause form.

3) Prove that "Is someone smiling?" using resolution technique. Draw the resolution tree.

Ans: (12M | Dec 2015)

(i) Represent these axioms in first order predicate logic.

Ans: Object Constant: x = people

Predicates: G(x) = People Graduating, H(x) = Happy People, S(x) = Smiling People

FOPL: $\forall x \; G(x) \Rightarrow H(x)$

$\forall x \; H(x) \Rightarrow S(x)$

$\exists x \; G(x)$ OR $\forall x \; G(f(x)) \approx G(y)$

(ii) Convert each formula to clause form.

Ans: **Rules(axioms) :** $\forall x \; G(x) \Rightarrow H(x)$; CNF: $\neg G(x) \vee H(x)$

$\forall x \; H(x) \Rightarrow S(x)$; CNF: $\neg H(x) \vee S(x)$

Fact: $\exists x \; G(x)$ Skolemized as G(y)

Clause Forms: $\forall x \; \neg$Graduating(x) v Happy(x)

$\forall x \; \neg$Happy(x) v Smiling(x)

$\exists x$ Graduating(x)

(iii)Prove that "is someone smiling?" using resolution technique. Draw the resolution tree.

Ans:

Goal : $\neg \exists x$ Smiling(x) (Negating the Conclusion)

$= \forall x \neg$ Smiling(x) //Reduce scope of negation

Resolution : Standardize variables apart

$\forall x \; \neg$Graduating(x) v Happy(x)

$\forall y \; \neg$Happy(y) v Smiling(y)

$\exists z$ Graduating(z) : Graduating(someone) // Eliminate $\exists$

$\forall w \neg$ Smiling(w)

Drop all $\forall$

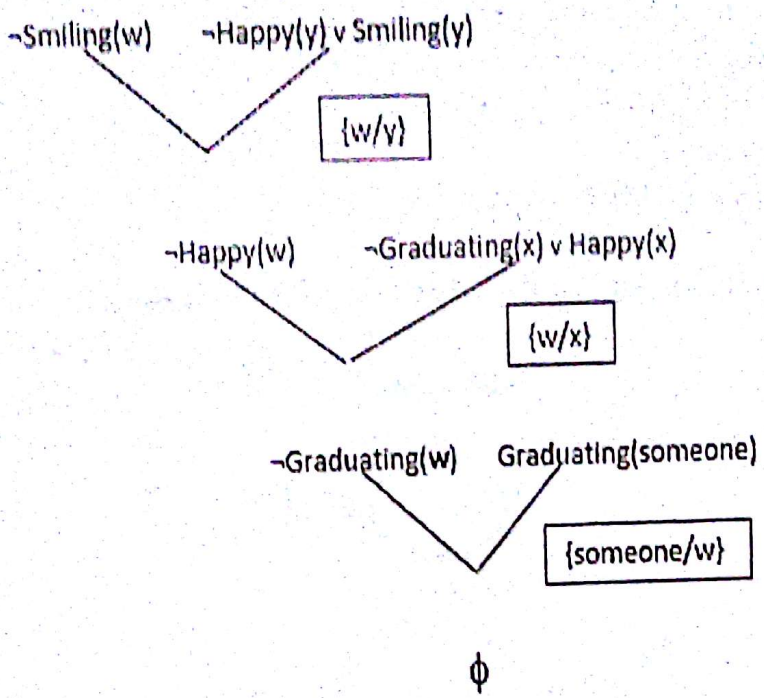
¬Graduating(x) v Happy(x)

¬Happy(y) v Smiling(y)

Graduating(someone)

¬ Smiling(w)

Resolution Tree:



ϕ

Hence proved Someone is smiling!

Q4) It is known that whether or not a person has cancer is directly influenced by whether she is exposed to second-hand smoke and whether she smokes. Both of these things are affected by whether her parents smoke. Cancer reduces a person's life expectancy.

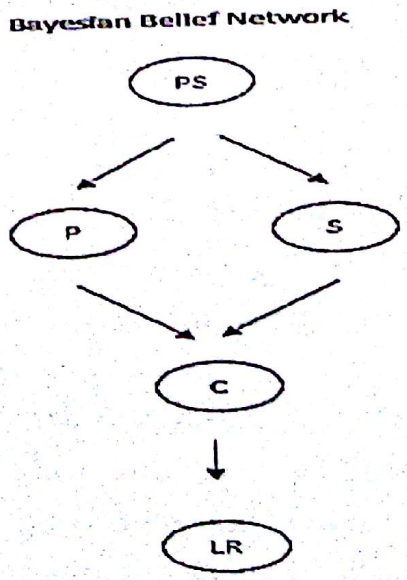
(i)Draw the Bayesian Belief Network for the above situation.

(ii)Associate a conditional probability table for each node.

(6M | Dec 2015)

Ans:

(i)Draw the Bayesian Belief Network for the above situation.

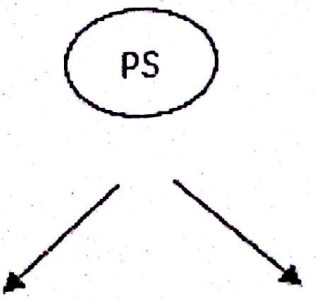


PS – Person's Parents Smoke
P – Person Smokes
S – Person exposed to Second-hand Smoke
C – Person has Cancer
LR – Person's Life Expectancy is reduced

(ii)Associate a conditional probability table for each node.

Ans:

P(PS=True)

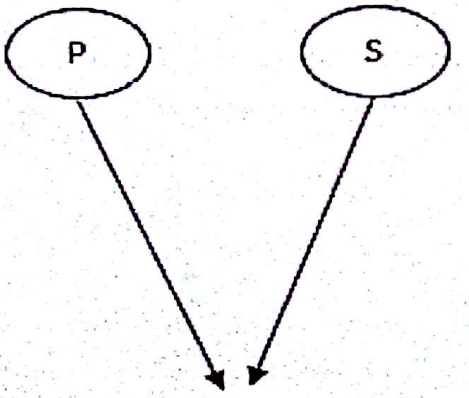


CPT (P)

PS	P(P=true | PS)
True	
False	

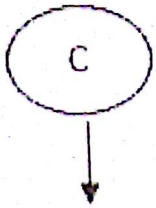
CPT (S)

PS	P(S=true | PS)
True	
False	



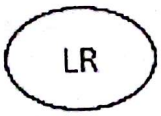
CPT (C)

PS	P(C=true | P, S)
True	True
False	True
True	False
False	False



CPT (LR)

C	P(LR=true | C)
True	
False	



Q5) Write a PROLOG program to find Fibonacci series. (10M | Dec 2015)

Ans:

```
domains
        x = integer
predicates
        fibonacci(x)
clauses
        fibonacci(1).
        fibonacci(N) :-
                N1 = N-1,
                N1 >= 0,!,
                fibonacci(N1),
                write(F1," ,"),
                F= F1+N.
```

Q6) Write first order logic statements for following statements:

(i) If a perfect square is divisible by a prime p then it is also divisible by square of p.

(ii) Every perfect square is divisible by some prime.

(iii) Alice does not like Chemistry and History.

(iv) If it is Saturday and warm, then San is in the park.

(v) Anything anyone eats and is not killed by is food.

Ans: **(10M | May 2016)**

(i) If a perfect square is divisible by a prime p then it is also divisible by square of p.

FOL : ∀x [∀p (PerfectSquare(x) ^ Divisible(x,p)) ⇒ Divisible(x,p^2)] // Prime 'p', Number 'x'

(ii) Every perfect square is divisible by some prime.

FOL : ∀x [∃p (PerfectSquare(x) ⇒ Divisible(x,p) ^ Prime(p)]

(iii) Alice does not like Chemistry and History.

FOL : Likes(Alice, Chemistry) ^ Likes(Alice, History)

(iv) If it is Saturday and warm, then San is in the park.

FOL : ∀x (Saturday(x) ^ Warm(x) ⇒ IN(San, Park)) //Day x

(v) Anything anyone eats and is not killed by is food.

FOL : ∀x [∀y (Eats(x, y) ^ Killed(y, x)) ⇒ ∀y Food(y)

Q7) Find the probabilistic inference by enumeration of entries in a full joint distribution table shown in figure 1.

(i) No cavity when toothache is there

(ii) p(Cavity! toothache or catch)

	toothache		¬toothache	
	catch	¬catch	catch	¬catch
cavity	.108	.012	.072	.008
¬cavity	.016	.064	.144	.576

Figure 1

Ans: **(10M | May 2016)**

(i) No cavity when toothache is there

P(¬cavity | toothache) = P(¬cavity^toothache) / P(toothache)

= (0.016 + 0.064) / (0.108 + 0.012 + 0.016 + 0.064)

= **0.4**

(ii) p(Cavity | toothache or catch)

= P(Cavity | toothache) + p(catch)

= [P(cavity^toothache) / P(toothache)] + P(catch)

= [(0.108+0.012)]/[(0.108+0.012+0.016+0.064 + (0.108+0.016+0.072+0.144)

$= 0.96$

Q8) Explain the steps involved in conveying the propositional logic statement into CNF with a suitable example. (10M | May 2016)

Ans:

CNF in first order logic

1. Eliminate implications $\Leftrightarrow, \Rightarrow$
2. Reduce the scope of $\neg$

» de Morgan's laws

$\neg(w1 \vee w2) \equiv \neg w1 \wedge \neg w2$

$\neg(w1 \wedge w2) \equiv \neg w1 \vee \neg w2$

» Elimination of repeated negations $(\neg \neg w \equiv w)$

» Combination of $\neg$ with quantifiers.

$\neg(\forall x)\ w(x) \equiv (\exists x)\ \neg w(x)$

$\neg(\exists x)\ w(x) \equiv (\forall x)\ \neg w(x)$

3. **Standardize variables:** Rename variables so that each different variable in the set of wffs has a different symbol

$:(\forall x)[P(x) \Rightarrow R(x)] \vee [(\exists x) P(x)] \equiv [(\forall x)[P(x) \Rightarrow R(x)]] \vee [(\exists y) P(y)]$

4. **Skolemization:** Eliminate existential quantifiers and replace existentially quantized variables by Skolem consants or Skolem functions as appropriate.

5. **Convert to prenex form** by moving all universal quantifiers to the beginning of the wff.

wff in prenex form = **Prefix** (string of quantifiers) + **Matrix** (quantifier-free formula)

6. **Drop universal quantifiers.**
7. Use distributive laws and equivalence rules of propositional logic to transform the matrix to CNF.

<u>Example Conversion to CNF</u>

$[(\forall x)\ Q(x)] \Rightarrow (\forall x)(\forall y)\ [(\exists z)\ [P(x,y,z) \Rightarrow (\forall u)\ R(x,y,u,z)]]$

1. Eliminate implications $\Leftrightarrow, \Rightarrow$

$\neg[(\forall x)\ Q(x)] \vee$

$(\forall x)(\forall y)\ [(\exists z)\ [\neg P(x,y,z) \vee (\forall u)\ R(x,y,u,z)]]$

2. Reduce scope of negations:

$[(\exists x)\ \neg Q(x)] \vee$
$(\forall x)(\forall y)\ [(\exists z)\ [\neg P(x,y,z) \vee (\forall u)\ R(x,y,u,z)]$

3. Standardize variables

$[(\exists w)\ \neg Q(w)] \vee$
$(\forall x)(\forall y)\ [(\exists z)\ [\neg P(x,y,z) \vee (\forall u)\ R(x,y,u,z)]$

4. Skolemization:

$\neg Q(A) \vee (\forall x,y)\ [\neg P(x,y,f(x,y)) \vee (\forall u) R(x,y,u,f(x,y))]$

5. Prenex form:

$(\forall x,y,u)\ [\neg Q(A) \vee \neg P(x,y,f(x,y)) \vee R(x,y,u,f(x,y))]$

6. Drop universal quantifiers

$\neg Q(A) \vee \neg P(x,y,f(x,y)) \vee R(x,y,u,f(x,y))$

7. Convert to CNF:

wff already in CNF.

Chap 5| Planning and Learning

Q1) Define heuristic function. Give an example heuristics function for Blocks World Problem.

(5M|Dec 2015)

Ans:

Heuristic Function is a function that estimates the cost of getting from one place to another (from the current state to the goal state.) Also called as simply a **heuristic.**

Used in a decision process to try to make the best choice of a list of possibilities (to choose the move more likely to lead to the goal state.) Best move is the one with the least cost.

It can also be defined thus as a function that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow.

Example for Blocks World Problem:

Consider below mentioned Start & Goal states for Blocks World

	A	C
	B	D
P	Q	R

Start State

A	C	
B	D	
P	Q	R

Goal State

Heuristics function:

h(s) = Number of places with incorrect block immediately on top of it

Q2) Find the heuristics value for a particular state of the Blocks World Problem.

(5M|Dec 2015)

Ans:

Consider below mentioned Blocks World Problem with 'X' as sample intermediate state

	A	C
	B	D
P	Q	R

Start State

	C	
B	D	A
P	Q	R

State X

A	C	
B	D	
P	Q	R

Goal State

To Find: h(X)=?
Solution:

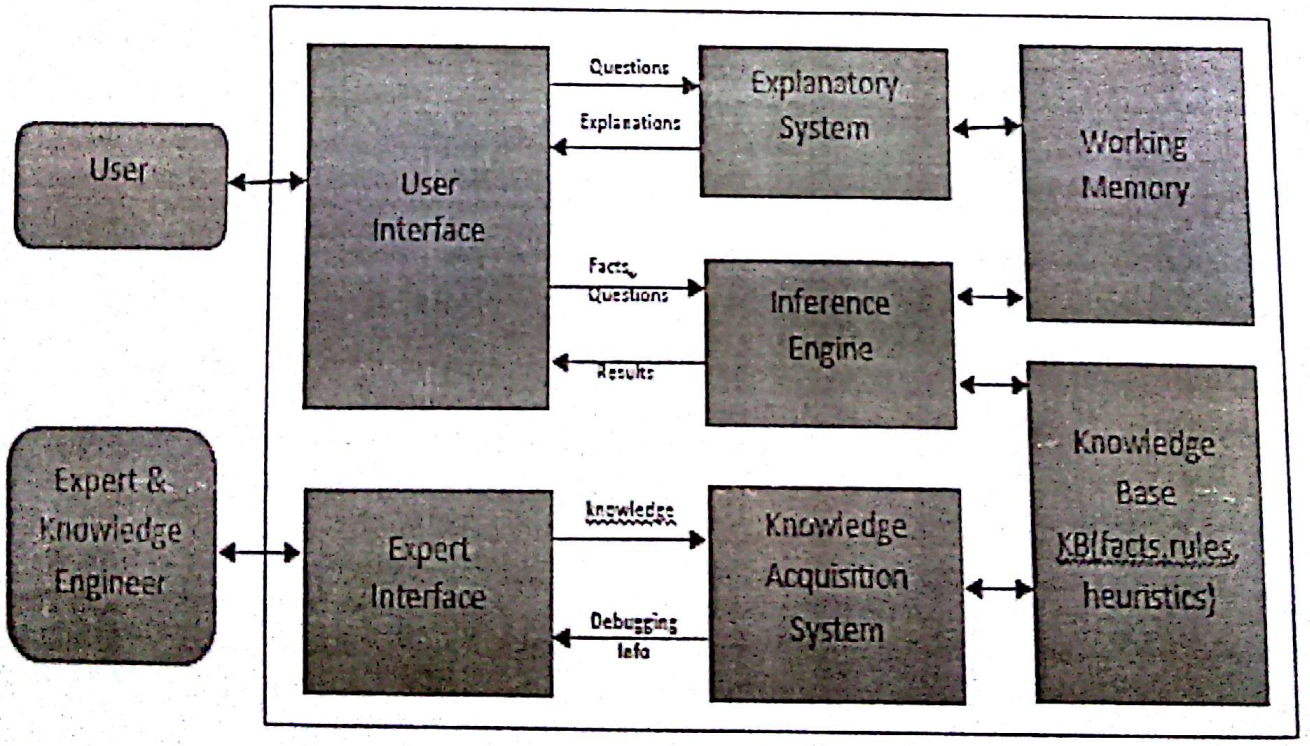
Blocks Arrangements =>

B P	Correct
C D	Correct
D Q	Correct
A R	Incorrect

Therefore, h(X) = 1

Q3) Draw and describe the architecture of expert system.
Ans: **(6M, 5M | Dec 2015, May 2016)**

Architecture of Expert System



The Architecture of an Expert System (ES) consists of the following major components:

- **Knowledge Base (KB):** repository of special heuristics or rules that direct the use of knowledge, facts (productions). It contains the knowledge necessary for understanding, formulating, & problem solving.
- **Working Memory(Blackboard):** if forward chaining used
  It describes the current problem & record intermediate results
  Records Intermediate Hypothesis & Decisions: 1. Plan, 2. Agenda, 3. Solution
- **Inference Engine:** the deduction system used to infer results from user input & KB
  It is the brain of the ES, the control structure(rule interpreter)
  It provides methodology for reasoning
- **Explanation Subsystem (Justifier):** Traces responsibility & explains the ES behaviour by interactively answering question: Why?, How?, What?, Where?, When?, Who?
- **User Interface:** interfaces with user through Natural Language Processing (NLP), or menus & graphics. Acts as Language Processor for friendly, problem-oriented communication

Shell = Inference Engine + User Interface

The Human Elements in ESs

- **Expert:** Has the special knowledge, judgement, experience and methods to give advice and solve problems. Provides knowledge about task performance
- **Knowledge Engineer:** Usually also the System Builder
  Helps the expert(s) structure the problem area by interpreting and integrating human answers to questions, drawing analogies, posing counter examples, and bringing to light conceptual difficulties.

The Expert & the knowledge Engineer should Anticipate Users' needs & Limitations when designing Expert Systems

- **User:** Possible Classes of Users can be
  - A non-expert client seeking direct advice (ES acts as a Consultant or Advisor)
  - A student who wants to learn (ES acts as an Instructor)
  - An ES builder improving or increasing the knowledge base(ES acts as a Partner)
  - An Expert (ES acts as a Colleague or an Assistant)

Q4) Consider a decision tree for the following set of samples. Write any two decision rules obtained from the tree. Classify a new sample with (gender = "Female"; height = "1.92m")

Person ID	Gender	Height	Class
1	Female	1.6m	Short
2	Male	2m	Tall
3	Female	1.9m	Medium
4	Female	2.1m	Tall
5	Female	1.7m	Short
6	Male	1.85m	Medium
7	Female	1.6m	Short
8	Male	1.7m	Short
9	Male	2.2m	Tall

Ans: **(10M | Dec 2015)**

P (short) = 4/9
P (medium) = 2/9
P (tall) = 3/9
Dividing the height attribute in to six ranges.
[0, 1.6], [1.6, 1.7], [1.7, 1.8], [1.8, 1.9], [1.9,2.0], [2.0, Infinity]
Gender has only two values Male and Female.
Total Number of short person = 4
Total Number of medium persons = 2
Total Number of tall persons = 3

Attribute	Value	Count			Probabilities		
		Short	Medium	Tall	Short	Medium	Tall
Gender	Male	1	1	2	1/4	1/2	2/3
	Female	3	1	1	1/4	1/2	1/3
Height	[0,1.6]	2	0	0	2/4	0	0
	[1.6, 1.7]	2	0	0	2/4	0	0
	[1.7, 1.8]	0	0	0	0	0	0
	[1.8, 1.9]	0	2	0	0	1	0
	[1.9, 2.0]	0	0	1	0	0	1/3
	[2.0, ∞]	0	0	2	0	0	2/3

Consider new sample as Tuple t = {Female, 1.92m}

P (t l Short) = 1/4*0=0
P (t l Medium) = 1/2*0= 0
P (t l Tall) =2/3 *1/3=0.22

Therefore likelihood of being Short = P (t l short) * P (short) = 0* 4/9=0
Similarly,
Likelihood of being Medium = P (t l Medium) * P (Medium) = 0*2/9=0
Likelihood of being Tall = P (t l Tall) * P (Tall) = 1/3 * 3/9=0.11
Then estimate P (t) by adding individual likelihood values since t will be short, medium or tall
P (t) = 0+0+0+0.11=0.11
Finally Actually Probability of each event
P (short l t) = (P (t l short) * P (short)) /P (t) = 0
Similarly,
P (medium l t) = (P (t l medium) * P (medium)) /P (t) = 0
P (Tall l t) = (P (t l Tall) * P (Tall)) /P (t) = **0.33**

Result : New Tuple is a **Tall** as it has the highest Probability.

Q5) Explain a partial order planner with an example.

Ans: **(6M | Dec 2015)**

Partial-Order Planner(POP) is a regression planner; it uses problem decomposition; it searches plan space rather than state space; it build partially-ordered plans; and it operates by the principle of least-commitment.

A plan in POP (whether it be a finished one or an unfinished one) comprises:

- A set of plan steps. Each of these is a STRIPS operator, but with the variables instantiated.
- A set of ordering constraints: Si -< Sj means step Si must occur sometime before Sj (not necessarily immediately before).
- A set of causal links: $Si \xrightarrow{c} Sj$ means step Si achieves precondition c of step Sj .

So, it comprises actions (steps) with constraints (for ordering and causality) on them.

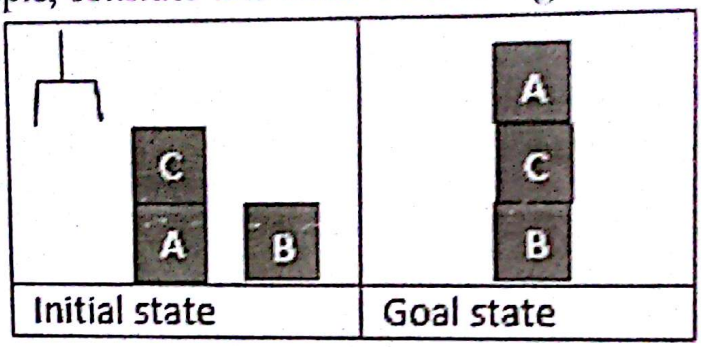
The algorithm needs to start off with an initial plan. This is an unfinished plan, which we will refine until we reach a solution plan.

The initial plan comprises two dummy steps, called Start and Finish.

Start is a step with no preconditions, only effects: the effects are the initial state of the world. Finish is a step with no effects, only preconditions: the preconditions

are the goal.

By way of an example, consider this initial state and goal state:



These would be represented in POP as the following initial plan:

```
Plan(STEPS: {S1: Op(ACTION: Start,
        EFFECT: clear(b) ^ clear(c) ^ on(c, a) ^
ontable(a) ^
          ontable(b) ^ armempty),
          S2: Op( ACTION: Finish,
        PRECOND: on(c, b) ^ on(a, c))},
ORDERINGS: {S1-< S2},
LINKS: {})
```

This initial plan is refined using POP's plan refinement operators. As we apply them, they will take us from an unfinished plan to a less and less unfinished plan, and ultimately to a solution plan. There are four operators, falling into two groups:

- Goal achievement operators

1. Step addition: Add a new step Si which has an effect c that can achieve an as yet unachieved precondition of an existing step Sj . Also add the following constraints: Si -<Sj and Si c Sj and Start -<Si -< Finish.
2. Use an effect c of an existing step Si to achieve an as yet unachieved precondition of another existing step Sj . And add just two constraints: Si -< Sj and Si c Sj.

- Causal links must be protected from threats, i.e. steps that delete (or negate or clobber) the protected condition. If S threatens link Si c Sj :

1. **Promote**: add the constraint S -< Si; or
2. **Demote**: add the constraint Sj -< S

The goal achievement operators ought to be obvious enough. They find preconditions of steps in the unfinished plan that are not yet achieved. The two goal achievement operators remedy this either by adding a new step whose effect achieves the precondition, or by exploiting one of the effects of a step that is already in the plan.

The promotion and demotion operators may be less clear. Why are these needed? POP uses problem-decomposition: faced with a conjunctive precondition, it uses goal achievement on each conjunct separately. But, as we know, this brings the risk that the steps we add when achieving one part of a precondition might interfere with the achievement of another precondition. And the idea of promotion and demotion is to add ordering constraints so that the step cannot interfere with the achievement of the precondition.

Finally, we have to be able to recognise when we have reached a solution plan: a *finished* plan.

A solution plan is one in which:

- every precondition of every step is achieved by the effect of some other step and all possible clobberers have been suitably demoted or promoted; and
- there are no contradictions in the ordering constraints, e.g. disallowed is Si -< Sj and Sj -<Si; also disallowed is Si -<Sj , Sj -< Sk and Sk -< Si.

Note that solutions may still be partially-ordered. This retains flexibility for as long as possible. Only immediately prior to execution will the plan need linearisation, i.e. the imposition of arbitrary ordering constraints on steps that are not yet ordered. (In fact, if there's more than one agent, or if there's a single agent but it is capable of multitasking, then some linearisation can be avoided: steps can be carried out in parallel.)

Q6) Explain decision tree learning with an example. What are decision rules? How to use it for classifying new samples?

Ans: (10M | May 2016)

Decision Tree: a decision tree consists of Nodes, Edges & Leaves

In **Decision Tree Learning**, a new example is classified by submitting it to a series of tests that determine the class label of the example. These tests are organized in a hierarchical structure called a *decision tree.*

The training examples are used for choosing appropriate tests in the decision tree. Typically, a tree is built from top to bottom, where tests that maximize the information gain about the classification are selected first.

To classify a sample:

1. Start at the root
2. Perform the test
3. Follow the edge corresponding to outcome
4. Go to 2 unless leaf
5. Predict that outcome associated with the leaf

Rule induction

- Decision rules can be generated from a decision tree
- Each path from the root of the tree to a leave is a rule that classifies a set of examples
- It is also possible to generate a set of rules without creating a decision tree
- These algorithms learn the rules sequentially and not all at once like in a decision tree
- Some of these algorithm can even learn first order logic rules
- The key point of these methods is how to learn the best rule given a set of examples
- One possibility is to use the idea from decision trees and search in the space of conjunctions
- We start with the empty rule and each step we select the best new conjunction for the rule using an heuristic (eg. entropy) and a greedy strategy (eg. keep the best)
- To avoid local optima, a more exhaustive search can be performed, for example using beam search and storing the k best rules The majority class of the examples selected is assigned as the rule Prediction

- There are also other formalisms of rules that can be used
  - Decision tables
  - Decision lists
  - Ripple down rules
  - Defaukt Rules

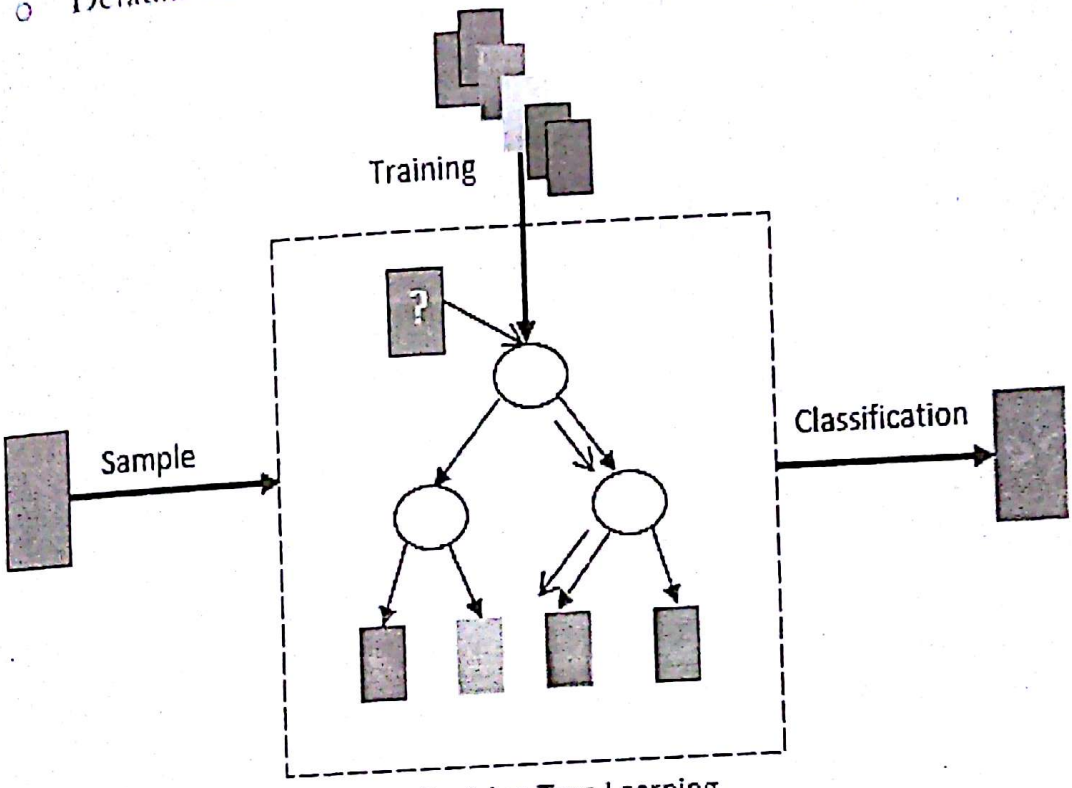
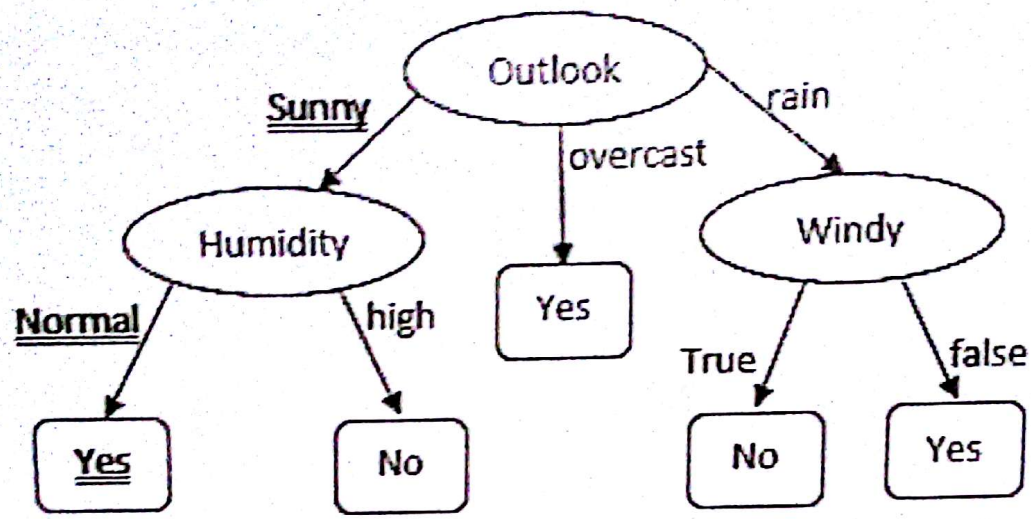


Fig. Decision Tree Learning

Application: A sample Task

Day	Temperature	Outlook	Humidity	Windy	Play Golf?
07-05	Hot	Sunny	High	Falsc	No
07-06	Hot	Sunny	High	True	No
07-07	Hot	Overcast	High	False	Yes
07-08	Cold	Sunny	Normal	False	Yes
07-09	Cold	Overcast	Normal	True	Yes
07-10	mild	Sunny	High	False	No
07-11	Cold	Sunny	Normal	False	Yes
07-12	mild	Rain	Normal	False	Yes
07-13	mild	Sunny	Normal	True	Yes
07-14	mild	Overcast	High	True	Yes
07-15	Hot	Overcast	Normal	False	Yes
07-16	mild	Rain	High	True	No
07-17	Cold	Rain	Normal	True	No

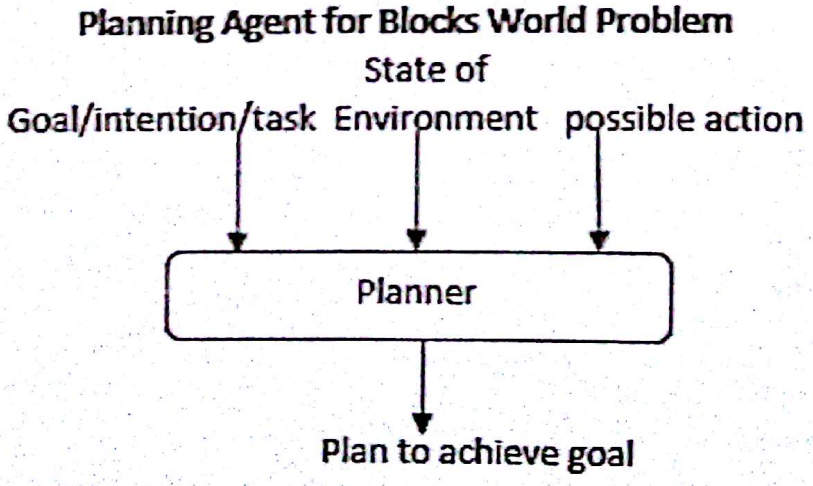
07-18	mild	Rain	High	False	Yes
Today	mild	Sunny	Normal	False	?



Decision: Play Golf Today!

Q7) Design a planning agent for a Blocks World problem. Assume suitable initial state and final state for the problem.

Ans: **(10M | May 2016)**



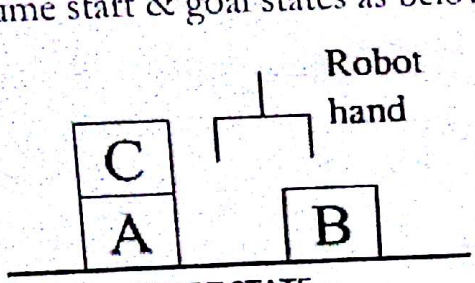
Designing the Agent

Idea is to give an *agent*:

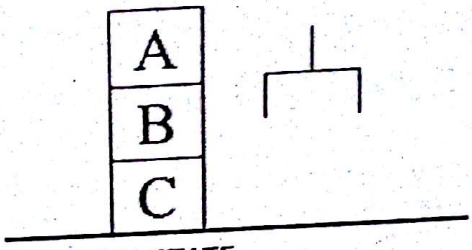
- Representation of goal/intention to achieve
- Representation of actions it can perform; and
- Representation of the environment;

Then have the agent generate a plan to achieve the goal.
The plan is generated entirely by the planning system, without human intervention.

Assume start & goal states as below:



START STATE
on(B, table)
on(A, table)
on(C, A)
hand empty
clear(C)
clear(B)



GOAL STATE
on(C, table)
on(B, C)
on(A, B)
hand empty
clear(A)

a. STRIPS : A planning system – Has rules with precondition deletion list and addition list

Sequence of actions :

b. Grab C
c. Pickup C
d. Place on table C
e. Grab B
f. Pickup B
g. Stack B on C
h. Grab A
i. Pickup A
j. Stack A on B

Rules:

k. *R1 : pickup(x)*

   a. Precondition & Deletion List : hand empty, on(x,table), clear(x)
   b. Add List : holding(x)

l. *R2 : putdown(x)*

   a. Precondition & Deletion List : holding(x)
   b. Add List : hand empty, on(x,table), clear(x)

m. R3 : *stack(x,y)*
   a. Precondition & Deletion List :holding(x), clear(y)
   b. Add List : on(x,y), clear(x)

n. R4 : *unstack(x,y)*
   a. Precondition & Deletion List : on(x,y), clear(x)
   b. Add List : holding(x), clear(y)

Plan for the assumed blocks world problem

For the given problem, Start → Goal can be achieved by the following sequence:

1. Unstack(C,A)
2. Putdown(C)
3. Pickup(B)
4. Stack(B,C)
5. Pickup(A)
6. Stack(A,B)

Chap 6| Applications

Q1) What are the levels of knowledge used in language understanding? Also write down the techniques used in NLP.
Ans: (10M | Dec 2015)

A natural language understanding system must have knowledge about what the words mean, how words combine to form sentences, how word meanings combine to from sentence meanings and so on.

The different **Levels (forms) of knowledge** required for natural language understanding are given below.

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.

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.

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.

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.

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.

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.

World Knowledge

Word 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.

There are several main techniques used in analysing natural language processing. Some of them can be briefly described as follows.

Pattern matching

The idea here is an approach to natural language processing is to interpret input utterances as a whole father than building up their interpretation by combining the structure and meaning of words or other lower level constituents. That means the interpretations are obtained by matching patterns of words against the input utterance. For a deep level of analysis in pattern matching a large number of patterns are required even for a restricted domain. This problem can be ameliorated by hierarchical pattern matching in which the input is gradually canonicalized through pattern matching against subphrases. Another way to reduce the number of patterns is by matching with semantic primitives instead of words.

Syntactically driven Parsing

Syntax means ways that words can fit together to form higher level units such as phrases, clauses and sentences. Therefore syntactically driven parsing means interpretation of larger groups of words are built up out of the interpretation of their syntactic constituent words or phrases. In a way this is the opposite of pattern matching as here the interpretation of the input is done as a whole.

Syntactic analyses are obtained by application of a grammar that determines what scentences are legal in the language that is being parsed.

Semantic Grammars

Natural language analysis based on semantic grammar is bit similar to syntactically driven parsing except that in semantic grammar the categories used are defined semantically and syntactically. There here semantic grammar is also involved.

Case frame instantiation

Case frame instantiation is one of the major parsing techniques under active research today. The has some very useful computational properties such as its recursive nature and its ability to combine bottom-up recognition of key constituents with top-down instantiation of less structured constituents.

Miscellaneous

Q1) Compare and Contrast problem solving agent and planning agent.

Ans: (5M | Dec 2015, May 2016)

	Problem Solving	Planning
1.	Problem solving is the process of diminution or abolishment of the divergence	Planning represents this process of finding out the necessary steps
2.	In contrast to a task, the necessary steps to take that transform the original state into the final state are not yet known and must still be figured out.	The steps are represented in a Plan.
3.	Problem solving involves planning as just a stage	Planning can be seen as a part of the problem solving process which is completed by the execution process.
4.	If problems during plan execution occur, the plan may be altered during run time.	Execution itself might not be a completely predefined process.
5.	DPS should enable the agents to work together in solving problems that are beyond a single agent's scope.	Distributed Planning activities involve a group of agents in the planning process.