Important Question & Answers (IQA)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SUBJECT CODE : CS6659

SUBJECT NAME : ARTIFICIAL INTELLIGENCE

Regulation: 2013 **Year and Semester: III/06**

ANNA UNIVERSITY, CHENNAI-25

REGULATION 2013

CS6659 ARTIFICIAL INTELLIGENCE L T P C

3 0 0 3

UNIT I INTRODUCTION TO AI AND PRODUCTION SYSTEMS 9

Introduction to AI-Problem formulation, Problem Definition -Production systems, Control strategies, Search strategies. Problem characteristics, Production system characteristics -Specialized production system- Problem solving methods - Problem graphs, Matching, Indexing and Heuristic functions -Hill Climbing-Depth first and Breath first, Constraints satisfaction - Related algorithms, Measure of performance and analysis of search algorithms.

UNIT II REPRESENTATION OF KNOWLEDGE 9

Game playing - Knowledge representation, Knowledge representation using Predicate logic, Introduction to predicate calculus, Resolution, Use of predicate calculus, Knowledge representation using other logic-Structured representation of knowledge.

UNIT III KNOWLEDGE INFERENCE 9

Knowledge representation -Production based system, Frame based system. Inference - Backward chaining, Forward chaining, Rule value approach, Fuzzy reasoning - Certainty factors, Bayesian Theory-Bayesian Network-Dempster - Shafer theory.

UNIT IV PLANNING AND MACHINE LEARNING 9

Basic plan generation systems - Strips -Advanced plan generation systems – K strips -Strategic explanations -Why, Why not and how explanations. Learning-Machine learning, adaptive Learning.

UNIT V EXPERT SYSTEMS 9

Expert systems - Architecture of expert systems, Roles of expert systems - Knowledge Acquisition –Meta knowledge, Heuristics. Typical expert systems - MYCIN, DART, XOON, Expert systems shells.

TOTAL : 45

TEXT BOOKS

1. Kevin Night and Elaine Rich, Nair B., “Artificial Intelligence (SIE)”, McGrawHill-2008. (Unit-1,2,4,5).
2. Dan W. Patterson, “Introduction to AI and ES”, Pearson Education, 2007. (Unit-III)

REFERENCES:

1. Peter Jackson, “Introduction to Expert Systems”, 3rd Edition, Pearson Education, 2007.
2. Stuart Russel and Peter Norvig “AI – A Modern Approach”, 2nd Edition, Pearson Education 2007.
3. Deepak Khemani “Artificial Intelligence”, Tata Mc Graw Hill Education 2013.
4. Nils.J.Nilsson “Principles of Atrificial Intelligence”, Springer –Verlag 1982.

TABLE OF CONTENTS

AIM AND OBJECTIVE OF THE SUBJECT

Aim of the subject

- To make machines do a variety of useful tasks.
- To design computer programs that mimics human behavior and expertise so that we can utilize human expertise even if expert is unavailable.
- To solve real world problems that can only be solved by human intelligence.

Objective of the subject

The student should be made to:

- Study the concepts of Artificial Intelligence.
- Learn the methods of solving problems using Artificial Intelligence.
- Introduce the concepts of Expert Systems and machine learning.

DETAILED LESSON PLAN

Text Book

1. S Kevin Night and Elaine Rich, Nair B., "Artificial Intelligence (SIE)", McGrawHill-2008. (Unit-1,2,4,5). **(Copies available in Library : Yes)**
2. Dan W. Patterson, "Introduction to AI and ES", Pearson Education, 2007. (Unit-III) **(Copies available in Library : Yes)**

REFERENCES:

1. Peter Jackson, "Introduction to Expert Systems", 3rd Edition, Pearson Education, 2007. **(Copies available in Library : Yes)**
2. Stuart Russel and Peter Norvig "AI – A Modern Approach", 2nd Edition, Pearson Education 2007. **(Copies available in Library : Yes)**
3. Deepak Khemani "Artificial Intelligence", Tata Mc Graw Hill Education 2013. **(Copies available in Library : No)**
4. Nils.J.Nilsson "Principles of Atrificial Intelligence", Springer –Verlag 1982. **(Copies available in Library : Yes)**
5. www.nptel.ac.in

Sl. No	Unit	Topic / Portions to be Covered	Hours Required / Planned	Cumulative Hrs	Books Referred
UNIT I - INTRODUCTION TO AI AND PRODUCTION SYSTEMS					
1	1	Introduction to AI-Problem formulation, Problem Definition	1	1	T1R3
2	1	Production systems, Control & Search strategies.	1	2	T1R3
3	1	Problem characteristics	1	3	T1R3
4	1	Production system characteristics	1	4	T1
5	1	Specialized production system	1	5	R4
6	1	Problem solving methods - Problem graphs, Matching, Indexing and Heuristic functions	1	6	T1R3
7	1	Hill Climbing	1	7	T1R3
8	1	Depth first and Breath first, Constraints satisfaction	1	8	T1R3
9	1	Related algorithms	1	9	T1

Sl. No	Unit	Topic / Portions to be Covered	Hours Required / Planned	Cumulative Hrs	Books Referred
10	1	Measure of performance and analysis of search algorithms.	1	10	T1
UNIT II – REPRESENTATION OF KNOWLEDGE					
11	2	Game playing	1	11	T1R3
12	2	Knowledge representation	1	12	T1T2
13	2	Knowledge representation using Predicate logic	1	13	T1
14	2	Introduction to predicate calculus	1	14	T1
15	2	Resolution	1	15	T1
16	2	Use of predicate calculus	1	16	T1
17	2	Knowledge representation using other logic	1	17	T1
18	2	Knowledge representation using other logic	1	18	T1
19	2	Structured representation of knowledge.	1	19	T1, R3
20	2	Structured representation of knowledge.	1	20	T1, R3
UNIT III- KNOWLEDGE INFERENCE					
21	3	Knowledge representation	1	21	T1
22	3	Production based system	1	22	T1
23	3	Frame based system	1	23	T1T2
24	3	Inference - Backward chaining	1	24	T1
25	3	Forward chaining	1	25	R3
26	3	Rule value approach, Fuzzy reasoning	1	26	T1R3
27	3	Certainty factors	1	27	T1T2
28	3	Bayesian Theory	1	28	T1T2
29	3	Bayesian Network	1	29	T1T2
30	3	Dempster - Shafer theory.	1	30	T1T2
UNIT IV – PLANNING AND MACHINE LEARNING					
31	4	Basic plan generation systems	1	31	T1R3 R4

Sl. No	Unit	Topic / Portions to be Covered	Hours Required / Planned	Cumulative Hrs	Books Referred
32	4	Strips	1	32	T1R3 R4
33	4	Advanced plan generation systems	1	33	T1R3 R4
34	4	R strips	1	34	T1R4
35	4	Strategic explanations	1	35	T1
36	4	Why, Why not and how explanations.	1	36	T1
37	4	Learning, Machine learning	1	37	T1R2
38	4	Adaptive Learning.	1	38	T1
UNIT V - EXPERT SYSTEMS					
39	5	Expert systems	1	39	T1
40	5	Architecture of expert systems	1	40	R2
41	5	Roles of expert systems	1	41	T1
42	5	Knowledge Acquisition, Meta knowledge	1	42	T1
43	5	Heuristics	1	43	R3
44	5	Typical expert systems - MYCIN	1	44	R3
45	5	DART	1	45	R3
46	5	XCON	1	46	R3
47	5	Expert systems shells.	1	47	T1

UNIT I INTRODUCTION TO AI AND PRODUCTION SYSTEMS

Introduction to AI-Problem formulation, Problem Definition -Production systems, Control strategies, Search strategies. Problem characteristics, Production system characteristics -Specialized production system- Problem solving methods - Problem graphs, Matching, Indexing and Heuristic functions -Hill Climbing-Depth first and Breath first, Constraints satisfaction - Related algorithms, Measure of performance and analysis of search algorithms.

PART A

1. List down the characteristics of intelligent agent. Apr/May 2011

1) The intelligent agent must learn and improve through interaction with the Environment.
2) The intelligent agent must adapt online and in the real time situation.
3) The intelligent agent must accommodate new problem solving rules incrementally.
4) The intelligent agent must have memory which must exhibit storage and retrieval capacities.

2. What do you mean by local maxima with respect to search technique? May-11

Local maximum is the peak that is higher than each of its neighbor states, but lower than the global maximum. i.e. a local maxima is a tiny hill on the surface whose peak is not as high as the main peak. Hill climbing fails to find optimum solution when it encounters local maxima. Any small moves from here also make things worse. At local maxima all the search procedure turns out to be wasted here. It is like a dead end.

3. What are the four components to define a problem? Define them. Apr/May 2013

The four components to define a problem are,

1. **Initial state**- It is the state in which agent starts in.
2. **A description of possible actions**- It is the description of possible actions which are available to the agent.
3. **The goal test**- It is the test determines whether a given state is goal state.
4. **A path cost function**- It is the function that assigns a numeric cost to each path. The problem solving agent is expected to choose a cost function that reflects its own performance measure.

4. What is the advantage of heuristic function? Dec-11

Heuristic function ranks alternative paths in various search algorithms at each branching step, based on the available information, so that a better path is chosen.

The main advantage of heuristic function is that it guides for which state to explore now, while searching. It makes use of problem specific knowledge like constraints to check the goodness of a state to be explored. This drastically reduces the required searching time.

5. What is AI?

Artificial Intelligence (AI) is a branch of *Science* which deals with helping machines finding solutions to complex problems in a more human-like fashion. This involves borrowing characteristics from human intelligence, and applying them as algorithms in a computer friendly way. This word was coined by McCarthy in the year 1956.

6. Define the term process operationalization.

The process of creating a formal description of the problem using the knowledge about the given problem, so as to create a program for solving a problem is called process operationalization. The steps are

- Define a state space.
- Identify the initial states.
- Specify goal states
- Specify set of rules.

7. Define Constraint satisfaction. (May 2012)

- It is a search procedure that operates in a space of constrained sets.
- The initial states contain the constraints that are given in problem description.
- A goal state is any state that has been constrained to a tolerable level, where the level of tolerance depends upon the problem.
- CS is a two-step process.
    i. Discover the constraints involved in the problem and propagate them throughout the system.
    ii. Any hypotheses that strengthen the constraints is formulated. If a solution is found, then we do nothing else, we backtrack and try a different guess.

8. List the various types of searching available in AI.

Uninformed searching

1. Breadth first search
2. Depth first search
3. Brute force or blind search
4. Greedy search

Informed searching or Heursitic search

1. Best first search
2. Branch and Bound Search
3. A* Search
4. AO* Search
5. Hill Climbing
6. Constraint satisfaction
7. Means end analysis

9. What are the various problem solving methods?

The various methods used are

- Problem graph
- Matching
  - Indexing
- Heuristic function

10. Define Production system.

A production system is one that consists of

- A set of rules.
- One or more knowledge/database.
- A control strategy
- A rule applier.

Production system also incorporates a family of general production system interpreters that include

I. Basic production system languages.

II. Expert System shells.

11. List the various classes of production system.

The various classes are

a. A Monotonic production system: A production system in which applications of a rule never prevents the future application of another rule that could also been applied at the time the first rule was selected.

b. A nonmonotonic production system: A production system in which the rule of the monotonic production system doesn't apply.

c. A partially commutative production system: A production system which has the property of transforming x into state y on the application of specific sequence of rules.

d. A commutative production system: A production system that is both monotonic and partly commutative.

12. What are the various building blocks to solve a problem?

The various things to be done to solve a problem is

a. The problem should be defined precisely.

b. Analyze the problem.

c. Isolate and represent the task knowledge that is necessary to solve the problem.

d. Choose the best problem solving technique and apply it to the particular problem.

13. What is an AI technique? (June 2013)

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

a. Generalization is captured through use of knowledge.

b. People who provide the knowledge should also understand it.

c. It is easily modifiable to correct errors so they refine changes in the world.

d. It is used in many situations that is not accurate or complete.

e. It narrows the range of possibilities.

14. What is alpha? (MAY/JUNE 2016)

A ridge which is an area In the search that is higher than the surroundings areas, but cannot be searched in a simple move.

15. How much knowledge would be required by a perfect program for the problem of playing chess? Assume that unlimited computing power is available.

The rules for determining legal moves and some simple control mechanism that implements an appropriate search problem. Additional knowledge about such things as good strategy and tactics could of course help considerably to constrain the search and speed up execution of the program.

PART- B

1. Explain about informed search strategies in detail. (MAY/JUNE 2016)

We can also call informed search as Heuristics search. It can be classified as below

- Best first search
- Branch and Bound Search
- A* Search
- AO* Search
- Hill Climbing
- Constraint satisfaction
- Means end analysis

Heuristic is a technique which makes our search algorithm more efficient. Some heuristics help to guide a search process without sacrificing any claim to completeness and some sacrificing it.

Heuristic is a problem specific knowledge that decreases expected search efforts. It is a technique which sometimes works but not always. Heuristic search algorithm uses information about the problem to help directing the path through the search space.

These searches uses some functions that estimate the cost from the current state to the goal presuming that such function is efficient. A heuristic function is a function that maps from problem state descriptions to measure of desirability usually represented as number.

The purpose of heuristic function is to guide the search process in the most profitable directions by suggesting which path to follow first when more than is available.

Generally heuristic incorporates domain knowledge to improve efficiency over blind search.

In AI heuristic has a general meaning and also a more specialized technical meaning. Generally a term heuristic is used for any advice that is effective but is not guaranteed to work in every case.

For example in case of travelling sales man (TSP) problem we are using a heuristic to calculate the nearest neighbour. Heuristic is a method that provides a better guess about the correct choice to make at any junction that would be achieved by random guessing.

This technique is useful in solving though problems which could not be solved in any other way. Solutions take an infinite time to compute.

Classifications of heuristic search.

Best First Search

Best first search is an instance of graph search algorithm in which a node is selected for expansion based o evaluation function f (n). Traditionally, the node which is the lowest evaluation is selected for the explanation because the evaluation measures distance to the goal. Best first search can be implemented within general search frame work via a priority queue, a data structure that will maintain the fringe in ascending order of f values.

This search algorithm serves as combination of depth first and breadth first search algorithm. Best first search algorithm is often referred greedy algorithm this is because they quickly attack the most desirable path as soon as its heuristic weight becomes the most desirable.

Concept:

Step 1: Traverse the root node

Step 2: Traverse any neighbor of the root node, that is maintaining a least distance from the root node and insert them in ascending order into the queue.

Step 3: Traverse any neighbor of neighbor of the root node, that is maintaining a least distance fromthe root node and insert them in ascending order into the queue

Step 4: This process will continue until we are getting the goal node

Algorithm:

Step 1: Place the starting node or root node into the queue.

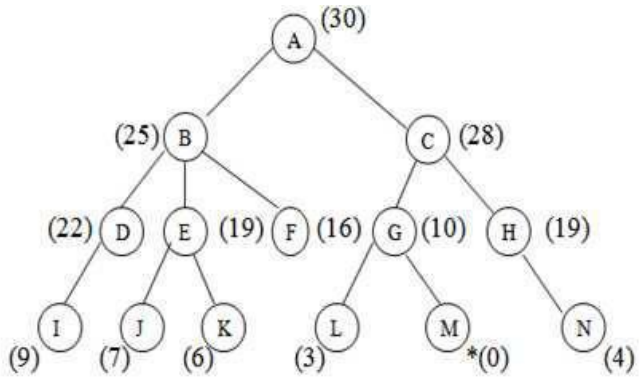
Step 2: If the queue is empty, then stop and return failure.

Step 3: If the first element of the queue is our goal node, then stop and return success.

Step 4: Else, remove the first element from the queue. Expand it and compute the estimated goal distancefor each child. Place the children in the queue in ascending order to the goal distance.

Step 5: Go to step-3

Implementation:



Step 1: Consider the node A as our root node. So the first element of the queue is A whish is not our goal node, so remove it from the queue and find its neighbor that are to inserted in ascending order. A

Step 2: The neighbors of A are B and C. They will be inserted into the queue in ascending order. B C A

Step 3: Now B is on the FRONT end of the queue. So calculate the neighbours of B that are maintaining a least distance from the roof. F E D C B

Step 4: Now the node F is on the FRONT end of the queue. But as it has no further children, so remove it from the queue and proceed further. E D C B

Step 5: Now E is the FRONT end. So the children of E are J and K. Insert them into the queue in ascending order.K J D C E

Step 6: Now K is on the FRONT end and as it has no further children, so remove it and proceed further J D C K

Step7: Also, J has no corresponding children. So remove it and proceed further.D C J

Step 8: Now D is on the FRONT end and calculates the children of D and put it into the queue. I C D

Step9: Now I is the FRONT node and it has no children. So proceed further after removing this node from the queue. C I

Step 10: Now C is the FRONT node .So calculate the neighbours of C that are to be inserted in ascending order into the queue.G H C

Step 11: Now remove G from the queue and calculate its neighbour that is to insert in ascending order into the queue. M L H G

Step12: Now M is the FRONT node of the queue which is our goal node. So stop here and exit. L H M

Advantage:

- It is more efficient than that of BFS and DFS.
- Time complexity of Best first search is much less than Breadth first search.
- The Best first search allows us to switch between paths by gaining the benefits of both breadth first and depth first search. Because, depth first is good because a solution can be found without computing all nodes and Breadth first search is good because it does not get trapped in dead ends.

Disadvantages:

- Sometimes, it covers more distance than our consideration.

Branch and Bound Search

Branch and Bound is an algorithmic technique which finds the optimal solution by keeping the best solution found so far. If partial solution can't improve on the best it is abandoned, by this method the number of nodes which are explored can also be reduced. It also deals with the optimization problems over a search that can be presented as the leaves of the search tree.

The usual technique for eliminating the sub trees from the search tree is called pruning. For Branch and Bound algorithm we will use stack data structure.

Concept:

Step 1: Traverse the root node.

Step 2: Traverse any neighbour of the root node that is maintaining least distance from the root node.

Step 3: Traverse any neighbour of the neighbour of the root node that is maintaining least distance fromthe root node.

Step 4: This process will continue until we are getting the goal node.

Algorithm:

Step 1: PUSH the root node into the stack.

Step 2: If stack is empty, then stop and return failure.

Step 3: If the top node of the stack is a goal node, then stop and return success.

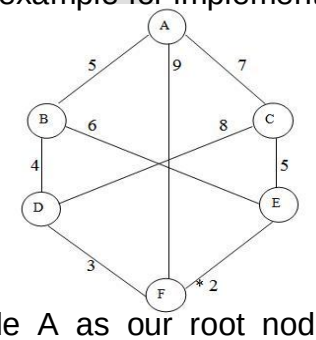
Step 4: Else POP the node from the stack. Process it and find all its successors. Find out the path containing all its successors as well as predecessors and then PUSH the successors which are belonging to the minimum or shortest path.

Step 5: Go to step 5.

Step 6: Exit.

Implementation:

Let us take the following example for implementing the Branch and Bound algorithm.



Step 1:

Consider the node A as our root node. Find its successors i.e. B, C, F. Calculate the distance from the root and PUSH them according to least distance.

A:Stating Node

B: 0+5 = 5 (The cost of A is 0 as it is the starting node)

F:0+9 = 9

C:0+7 = 7

Here B (5) is the least distance.

AB

Step 2:

Now the stack will be

C F B A

As B is on the top of the stack so calculate the neighbors of B.

D:0+5+4 = 9

E: 0+5+6 = 11

The least distance is D from B. So it will be on the top of the stack.

A 5 B 4 C

Step 3:

As the top of the stack is D. So calculate neighbours of D.

C F D B

C: 0+5+4+8 = 17

F: 0+5+4+3 = 12

The least distance is F from D and it is our goal node. So stop and return success.

Step 4:

C F D

Hence the searching path will be A-B -D-F

Advantages:

As it finds the minimum path instead of finding the minimum successor so there should not be any repetition. The time complexity is less compared to other algorithms.

Disadvantages:

The load balancing aspects for Branch and Bound algorithm make it parallelization difficult.

The Branch and Bound algorithm is limited to small size network. In the problem of large networks, where the solution search space grows exponentially with the scale of the network, the approach becomes relatively prohibitive.

A* SEARCH

A* is a cornerstone name of many AI systems and has been used since it was developed in 1968 by Peter Hart; Nils Nilsson and Bertram Raphael. It is the combination of Dijkstra's algorithm and Best first search. It can be used to solve many kinds of problems. A* search finds the shortest path through a search space to goal state using heuristic function. This technique finds minimal cost solutions and is directed to a goal state called A* search.

In A*, the * is written for optimality purpose. The A* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state.

This algorithm is complete if the branching factor is finite and every action has fixed cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. It can be defined by following formula

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

Where **g (n): The actual cost path from the start state to the current state.**

h (n): The actual cost path from the current state to goal state.

f (n): The actual cost path from the start state to the goal state.

For the implementation of A* algorithm we will use two arrays namely OPEN and CLOSE.

OPEN:

An array which contains the nodes that has been generated but has not been yet examined.

CLOSE:

An array which contains the nodes that have been examined.

Algorithm:

Step 1: Place the starting node into OPEN and find its f (n) value.

Step 2: Remove the node from OPEN, having smallest f (n) value. If it is a goal node then stop and return success.

Step 3: Else remove the node from OPEN, find all its successors.

Step 4: Find the f (n) value of all successors; place them into OPEN and place the removed node into CLOSE.

Step 5: Go to Step-2.

Step 6: Exit.

Implementation:

The implementation of A* algorithm is 8-puzzle game.

Advantages:

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

Disadvantages:

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

AO* Search: (And-Or) Graph

The Depth first search and Breadth first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined, since all goals following an AND nodes must be realized; where as a single goal node following an OR node will do. So for this purpose we are using AO* algorithm.Like A* algorithm here we will use two arrays and one heuristic function.

OPEN:

It contains the nodes that has been traversed but yet not been marked solvable or unsolvable.

CLOSE:

It contains the nodes that have already been processed.

Algorithm:

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in CLOSE

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved.If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable,then return failure and exit.

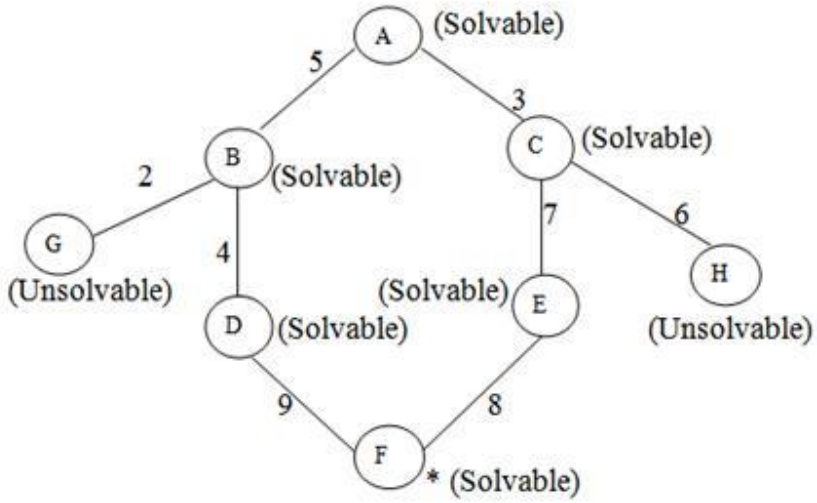
Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit.

Implementation:

Let us take the following example to implement the AO* algorithm.



Step 1:

In the above graph, the solvable nodes are A, B, C, D, E, F and the unsolvable nodes are G, H. Take A as the starting node. So place A into OPEN.

i.e. OPEN = A CLOSE = (NULL)

f A

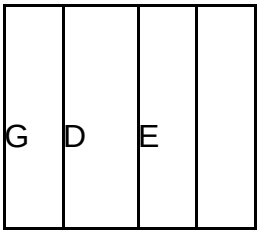
Step 2:

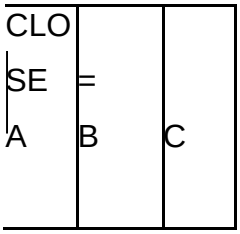
The children of A are B and C which are solvable. So place them into OPEN and place A into the CLOSE.

i.e. OPEN = B C CLOSE = A

Step 3:

Now process the nodes B and C. The children of B and C are to be placed into OPEN. Also remove B and C from OPEN and place them into CLOSE.

So OPEN = 

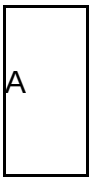


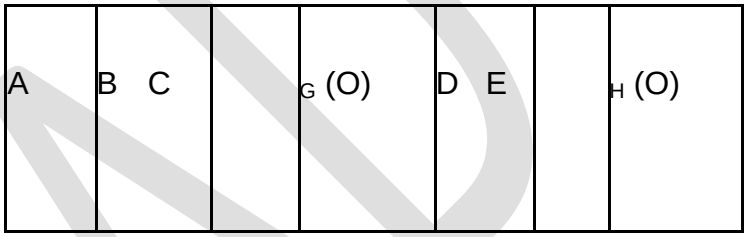
'O' indicated that the nodes G and H are unsolvable .

Step 4:

As the nodes G and H are unsolvable, so place them into CLOSE directly and process the nodes D and E.

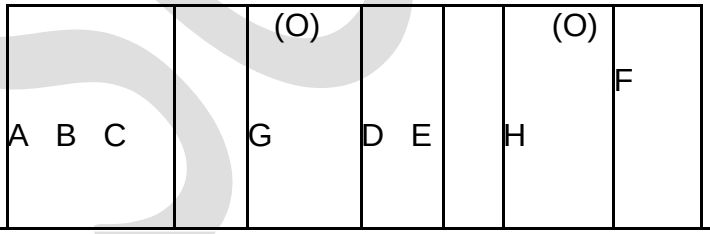
i.e. OPEN = CLOSE =





Step 5:

Now we have been reached at our goal state. So place F into CLOSE.

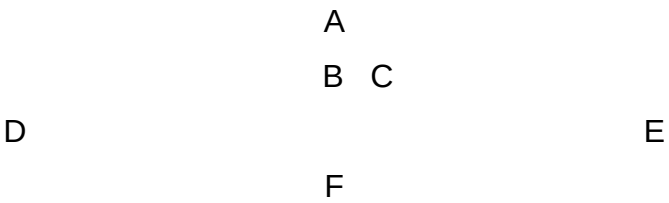


i.e. CLOSE =

Step 6:

Success and Exit

AO* Graph:



Advantages:

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

.**Disadvantages:**

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

2. Explain about uninformed search strategies in detail.

- Uninformed searching is classified as
  - Breadth first search
  - Depth first search
  - Brute force or blind search
  - Greedy search

Breadth First Search (BFS)

Breadth first search is a general technique of traversing a graph. Breadth first search may use more memory but will always find the shortest path first. In this type of search the state space is represented in form of a tree. The solution is obtained by traversing through the tree. The nodes of the tree represent the start value or starting state, various intermediate states and the final state. In this search a queue data structure is used and it is level by level traversal.

Breadth first search expands nodes in order of their distance from the root. It is a path finding algorithm that is capable of always finding the solution if one exists. The solution which is found is always the optional solution. This task is completed in a very memory intensive manner. Each node in the search tree is expanded in a breadth wise at each level.

Concept:

Step 1: Traverse the root node

Step 2: Traverse all neighbours of root node.

Step 3: Traverse all neighbours of neighbours of the root node.

Step 4: This process will continue until we are getting the goal node.

Algorithm:

Step 1: Place the root node inside the queue.

Step 2: If the queue is empty then stops and return failure.

Step 3: If the FRONT node of the queue is a goal node then stop and return success.

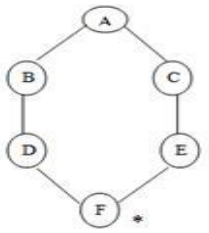
Step 4: Remove the FRONT node from the queue. Process it and find all its neighbours that are in readystate then place them inside the queue in any order.

Step 5: Go to Step 3.

Step 6: Exit.

Implementation:

Let us implement the above algorithm of BFS by taking the following suitable example.



Consider the graph in which let us take A as the starting node and F as the goal node (*)

Step 1: Place the root node inside the queue i.e. A

Step 2: Now the queue is not empty and also the FRONT node i.e. A is not our goal node. So move to step 3.

Step 3: So remove the FRONT node from the queue i.e. A and find the neighbour of A i.e. B and C B C A

Step 4: Now b is the FRONT node of the queue .So process B and finds the neighbours of B i.e. D. C D B

Step 5: Now find out the neighbours of C i.e. E

D B E

Step 6: Next find out the neighbours of D as D is the FRONT node of the queue

E F D

Step 7: Now E is the front node of the queue. So the neighbour of E is F which is our goal node. F E

Step 8: Finally F is our goal node which is the FRONT of the queue. So exit. F

Advantages:

- In this procedure at any way it will find the goal.
- It does not follow a single unfruitful path for a long time. It finds the minimal solution in case of multiple paths.

Disadvantages:

- BFS consumes large memory space. Its time complexity is more.
- It has long pathways, when all paths to a destination are on approximately the same search depth.

Depth First Search (DFS)

DFS is also an important type of uniform search. DFS visits all the vertices in the graph. This type of algorithm always chooses to go deeper into the graph. After DFS visited all the reachable vertices from a particular sources vertices it chooses one of the remaining undiscovered vertices and continues the search.

DFS reminds the space limitation of breath first search by always generating next a child of the deepest unexpanded nodded. The data structure stack or last in first out (LIFO) is used for DFS. One interesting property of DFS is that, the discover and finish time of each vertex from a parenthesis structure. If we use one open parenthesis when a vertex is finished then the result is properly nested set of parenthesis.

Concept:

Step 1: Traverse the root node.

Step 2: Traverse any neighbour of the root node.

Step 3: Traverse any neighbour of neighbour of the root node.

Step 4: This process will continue until we are getting the goal node.

Algorithm:

Step 1: PUSH the starting node into the stack.

Step 2: If the stack is empty then stop and return failure.

Step 3: If the top node of the stack is the goal node, then stop and return success.

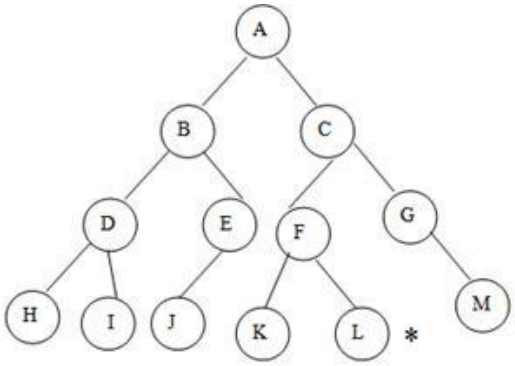
Step 4: Else POP the top node from the stack and process it. Find all its neighbours that are in ready stateand PUSH them into the stack in any order.

Step 5: Go to step 3.

Step 6: Exit.

Implementation:

Let us take an example for implementing the above DFS algorithm.



Examples of DFS

Consider A as the root node and L as the goal node in the graph figure

Step 1: PUSH the starting node into the stack i.e.A

Step 2: Now the stack is not empty and A is not our goal node. Hence move to next step.

Step 3: POP the top node from the stack i.e. A and find the neighbours of A i.e. B and C.

B C A

Step 4: Now C is top node of the stack. Find its neighbours i.e. F and G.

B F G C

Step 5: Now G is the top node of the stack. Find its neighbour i.e. M

B F M G

Step 6: Now M is the top node and find its neighbour, but there is no neighbours of M in the graph soPOP it from the stack. B F M

Step 7: Now F is the top node and its neighbours are K and L. so PUSH them on to the stack. B K L F

Step 8: Now L is the top node of the stack, which is our goal node.

B K L

Advantages:

- DFSconsumes very less memory space.
- It will reach at the goal node in a less time period than BFS if it traverses in a right path.
- It may find a solution without examining much of search because we may get the desired solution in the very first go

Disadvantages:

- It is possible that may states keep reoccurring. There is no guarantee of finding the goal node.
- Sometimes the states may also enter into infinite loops

.**Difference between BFS and DFS**

BFS

It uses the data structure queue.BFS is complete because it finds the solution if one exists

BFS takes more space i.e. equivalent to o ($\mathbf{b^0}$) where b is the maximum breath exist in a search tree and d is the maximum depth exit in a search tree. In case of several goals, it finds the best one.

DFS

It uses the data structure stack. It is not complete because it may take infinite loop to reach at the goal node. The space complexity is O (d).In case of several goals, it will terminate the solution in any order.

Greedy Search

This algorithm uses an approach which is quite similar to the best first search algorithm. It is a simple best first search which reduces the estimated cost of reach the goal. Basically it takes the closest node that appears to be closest to the goal. This search starts with the initial matrix and makes very single possible changes then looks at the change it made to the score.

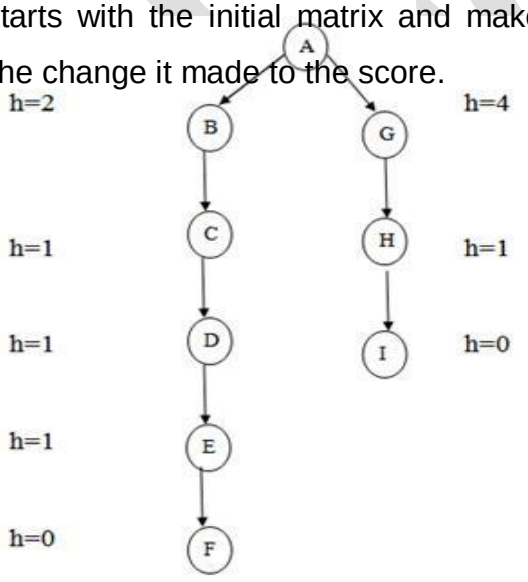


Figure Greedy Search

This search then applies the change till the greatest improvement. The search continues until no further improvement can be made. The greedy search never

makes never makes a lateral move .It uses minimal estimated cost h (n) to the goal state as measure which decreases the search time but the algorithm is neither complete nor optimal.

The main advantage of this search is that it is simple and finds solution quickly. The disadvantages are that it is not optimal, susceptible to false start.

3. Explain briefly the various problem characteristics.

A problem may have different aspects of representation and explanation. In order to choose the most appropriate method for a particular problem, it is necessary to analyze the problem along several key dimensions.

Some of the main key features of a problem are given below.

- Is the problem decomposable into set of independent smaller or easier sub problems?
- Can the solution step be ignored or undone?
- Is the problem universally predictable?
- Is a good solution to the problem obvious without comparison to all the possible solutions?
- Is the knowledge base to be used for solving the problem internally consistent?
- Is a large amount of knowledge absolutely required to solve the problem?
- Will the solution of the problem required interaction between the computer and the person?

The above characteristics of a problem are called as 7-problem characteristics under which the solution must take place.

Is the problem decomposable?

- Can the problem be broken down to smaller problems to be solved independently?
- Decomposable problem can be solved easily.

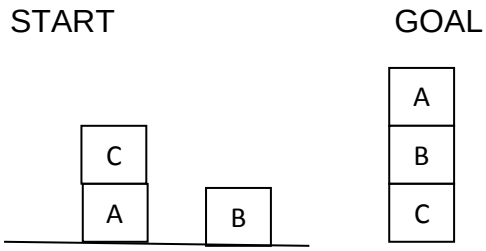
$$\int(x^2 + 3x + \sin^2 x.\cos^2 x)dx$$

$$\int x^2 dx \qquad \int 3xdx \qquad \int \sin^2 x.\cos^2 x dx$$

$$\int(1 - \cos^2 x)\cos^2 x dx$$

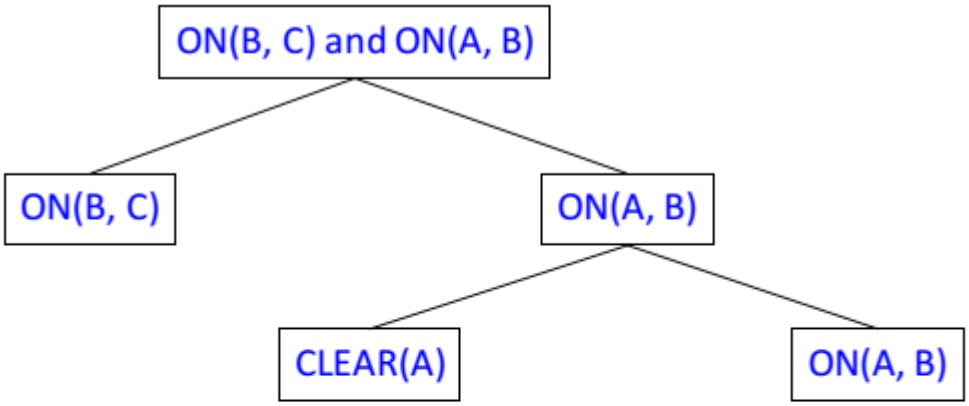
$$\int \cos^2 x dx \qquad -\int \cos^4 x dx$$

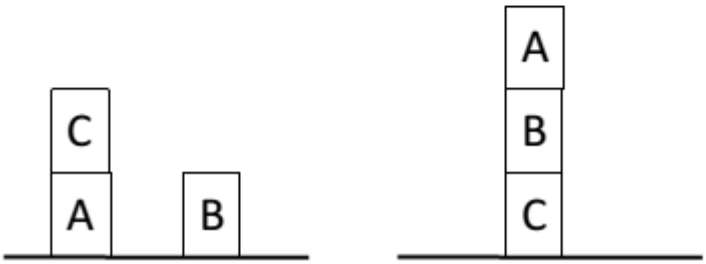
The next problem is called blocks world.



Blocks World

Applying the technique of problem decomposition to this simple blocks lead to the solution to the problem.





Decomposable problems can be solved by the Divide and Conquer technique of problem decomposition.

Can solution steps be ignored or undone?

Consider a mathematical theorem. First we take a lemma and prove it thinking that it is useful. But later we realize that the lemma is no help. But here there is no problem. The rules that could have been applied at the outset can still be applied. We can just proceed from the first place.

Example :

The 8 Puzzle. It is a square tray in which are placed 8 square tiles. The remaining 9th square is uncovered. Each tile has a number in it. A tile that is adjacent to the blank space can be slid into that space.

A game consisting 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.

Start state

2	8	3
1	6	4
7		5

Goal state.

1	2	3
8		4
7	6	5

We can backtrack and undo the first move

Mistakes can be recovered.

The control mechanism must keep track of order in which operations are performed so that the operations can be undone one at a time if necessary. The control structure for a theorem prover does not need to record all that information. Consider another problem of playing chess. Suppose a chess playing program realizes a stupid move after some moves, it cannot backup and start the game over from that point.

It can try a best of the current situation and go on from there. These problems - theorem proving, 8 puzzle and chess illustrate the difference between three important classes of problems. Ignorable problems can be solved using a simple control structure that never backtracks.

- Recoverable problems can be solved using backtracking.
- Irrecoverable problems can be solved by recoverable style methods via planning.

These three definitions make reference to the steps of the solution to a problem and thus may appear to characterize particular productive systems for solving a problem rather than the problem itself.

A different formulation of the same problem would lead to the problem being characterized differently.

The recoverability of a problem plays an important role in determining the complexity of the control structure necessary for its solution.

Ignorable problems can be solved using a simple control structure that never backtracks. such a control structure is easy to implement. Recoverable problems can be solved by a simple technique called backtracking. It can be implemented by using push down stack., in which the decisions are recorded in case they need later to be undone.

Irrecoverable problems will need to be solved by a system that expends a great deal of effort making each decision, since the decision must be final. Some irrecoverable problems can be solved by recoverable style methods used in a planning process in which an entire sequence of steps is analysed in advance to discover where it will lead before the first step is actually taken.

Is the universe predictable?

Consider the 8-Puzzle problem(Certain outcome). Every time we make a move, we know exactly what will happen. This means it is possible to plan the entire sequence of moves and be confident of the resulting state.

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

Consider another game of Playing Bridge(Uncertain outcome). We cannot know exactly where all the cards are or what the other players will do on their turns. The best is to investigate several plans and use probabilities of the various outcomes to choose a plan that has a highest expected probability of leading to a good score.

Generally,

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

Is a good solution absolute or relative?

Consider the problem of answering questions based on a database of simple facts.

1. Marcus was a man.
2. Marcus was a Pompeian.
3. Marcus was born in 40 A.D.

4. All men are mortal.
5. All Pompeian's died when the volcano erupted in 79 A.D.
6. No mortal lives longer than 150 years.
7. It is now 2004 A.D.

The question asked is " is Marcus alive"?

Different reasoning paths lead to the answer. It does not matter which path we follow.

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 1983.	axiom 7
9 Marcus age is 1943 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 1983	axiom 7
5 all Pompeian's died in 79	axiom 5
11 all Pompeian's are dead now	7,5
2 Marcus was a Pompeian	axiom 2
12 Marcus is dead	11,2

The Travelling Salesman Problem

The goal is to find the shortest route that visits each city exactly once.

Generally

- Any-path problems can be solved in a reasonable amount of time using heuristics that suggest good

  paths to explore. If the heuristics are not perfect , the search for a solution may not be as direct as possible.
- For best-path problems, much more exhaustive search will be performed.

  Is the solution a state or a path?

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

- "bank" refers to a financial situation or to a side of a river?
- "dish" or "pasta salad" was eaten?
- Does "pasta salad" contain pasta, as "dog food" does not contain "dog"?
- Which part of the sentence does "with the fork" modify?
- What if "with vegetables" is there?

No record of the processing is necessary.

The Water Jug Problem **:** The path that leads to the goal must be reported.

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

What is the role of knowledge?

Suppose if we have unlimited computing power available , then knowledge can be acquired easily to speed up the execution of the program.

Playing Chess

Knowledge is important only to constrain the search for a solution.

Reading Newspaper

Knowledge is required even to be able to recognize a solution.

Does the task require human-interaction?

It is useful for computers to be programmed to solve problems in ways that the majority of people woul not be able to understand. It is possible when human computer interaction is good.

There are two types of problems.

- Solitary problem, in which there is no intermediate communication and no demand for an explanation of the reasoning process.
- Conversational problem, in which intermediate communication is to provide either additional assistance to the computer or additional information to the user.

Example

Chess

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

4. **Consider the following problem:**

A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallonone. Neither jug has any measuring markings on it. There is pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water in the 4-gallon jug? Explicit Assumption: A Jug can e filled from the pump , water can be poured out of a jug onto the ground, water can be poured from one jug to another and that there are no other measuring devices available. (MAY/JUNE 2016)

State Representation and Initial State- we will represent a state of the problem as a tuple (x, y) where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug.

Note: 0<= x <=4, and 0<=y<=3.

Our initial state: (0,0)

Goal Predicate- state = (2,y) where 0<=y<=3.

Operators – we must define a set of operators that will take us from one state to another:

1. Fill 4-gal jug	(x,y) $x < 4$	→	(4,y)
2. Fill 3-gal jug	(x,y) $y < 3$	→	(x,3)
3. Empty 4-gal jug on ground	(x,y) $x > 0$	→	(0,y)
4. Empty 3-gal jug on ground	(x,y) $y > 0$	→	(x,0)
5. Pour water from 3-gal jug to fill 4-gal jug	(x,y) $0 < x+y \geq 4$ and $y > 0$	→	(4, y - (4 - x))
6. Pour water from 4-gal jug to fill 3-gal-jug	(x,y) $0 < x+y \geq 3$ and $x > 0$	→	(x - (3-y), 3)
7. Pour all of water from 3-gal jug into 4-gal jug	(x,y) $0 < x+y \leq 4$ and $y \geq 0$	→	(x+y, 0)
8. Pour all of water from 4-gal jug into 3-gal jug	(x,y) $0 < x+y \leq 3$ and $x \geq 0$	→	(0, x+y)

Through Graph Search, the following solution is found :

Gals in 4-gal jug	Gals in 3-gal jug	Rule Applied
0	0	
		1. Fill 4
4	0	
		6. Pour 4 into 3 to fill
1	3	
		4. Empty 3
1	0	
		8. Pour all of 4 into 3
0	1	
		1. Fill 4
4	1	
		6. Pour into 3
2	3	

5. Write in detail about the constraint satisfaction procedure with example.

A constraint search does not refer to any specific search algorithm but to a layer of complexity added to existing algorithms that limit the possible solution set. Heuristic and acquired knowledge can be combined to produce the desired result a constraint satisfaction problem is a special kind of search problem in which states are defined by the values of a set of variables and the goal state specifies a set of constraints that the value must obey.

There are many problems in AI in which the goal state is not specified in the problem and it requires to be discovered according to some specific constraint. Examples of some constraint satisfaction search include design problem, labeling graphs, robot path planning and cryptarithmetic problem etc.

Algorithm:

- Open all objects that must be assigned values in a complete solution.
- Repeat until all objects assigned valid values.
- Select an object and strengthen as much as possible. The set of constraints that

  apply to object.
- If set of constraints is different from previous set then open all objects that share any

  of these constraints. Remove selected objects.
- If union of constraints discovered above defines a solution, return solution.

- If union of constraints discovered above defines a contradiction, return failure.
- Make a guess in order to proceed. Repeat until a solution is found.
- Select an object with a number assigned value and try strengthening its constraints.

Consider the problem of solving a puzzle

E+V+O+L=AI, If A = 3 && 5<L<O<V<E<10, Find the value of I?

Many AI problems can be viewed as problems of constraint satisfaction.

As compared with a straightforward search procedure, viewing a problem as one of constraint satisfaction can reduce substantially the amount of search.

Constraint Satisfaction

- Operates in a space of constraint sets.
- Initial state contains the original constraints given in the problem.
- A goal state is any state that has been constrained "enough".

Two-step process:

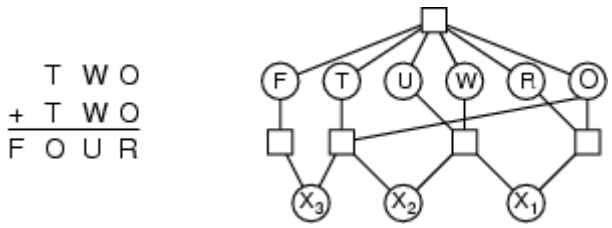
- Constraints are discovered and propagated as far as possible.
- If there is still not a solution, then search begins, adding new constraints.

Two kinds of rules:

- Rules that define valid constraint propagation.
- Rules that suggest guesses when necessary.

Constraints

- The simplest type is the **unary constraint**, which constraints the values of just one variable.
- A binary constraint relates two variables.
- Higher-order constraints involve three or more variables. Cryptarithmetic puzzles are an example:



Cryptarithmetic puzzles

•Variables:*F, T, U, W, R, O, X1, X2, X3*

•Domains: {*0,1,2,3,4,5,6,7,8,9*}

•Constraints:

–Alldiff(F,T,U,W,R,O)

–$O + O = R + 10 \cdot X1$

–$X1 + W + W = U + 10 \cdot X2$

–$X2 + T + T = O + 10 \cdot X3$

–$X3 = F$, $T \neq 0$, $F \neq 0$

```
  T W O
+ T W O
-------
F O U R
```

If 2+2 = 4, F=1, then how many fours ?

928+928=1856

867+867=1734

846+846=1692

836+836=1672

765+765=1530

734+734=1468

TWO

TWO

FOUR

938+938=1876

6. What are the problems encountered during hill climbing and what are the ways available to deal with these problems? / Write an algorithm for generate and test and simple hill climbing. (MAY/JUNE 2016)

Hill climbing search algorithm is simply a loop that continuously moves in the direction of increasing value. It stops when it reaches a “peak” where no n eighbour has higher value. This algorithm is considered to be one of the simplest procedures for implementing heuristic search. The hill climbing comes from that idea if you are trying to find the top of the hill and you go up direction from where ever you are. This heuristic combines the advantages of both depth first and breadth first searches into a single method.

The name hill climbing is derived from simulating the situation of a person climbing the hill. The person will try to move forward in the direction of at the top of the hill.

His movement stops when it reaches at the peak of hill and no peak has higher value of heuristic function than this. Hill climbing uses knowledge about the local terrain, providing a very useful and effective heuristic for eliminating much of the unproductive search space. It is a branch by a local evaluation function. The hill climbing is a variant of generate and test in which direction the search should proceed. At each point in the search path, a successor node that appears to reach for exploration.

Algorithm:

Step 1: Evaluate the starting state. If it is a goal state then stop and return success.

Step 2: Else, continue with the starting state as considering it as a current state.

Step 3: Continue step-4 until a solution is found i.e. until there are no new states left to be applied in the current state.

Step 4**:**

- Select a state that has not been yet applied to the current state and apply it to produce a new state.
- Procedure to evaluate a new state.
  - If the current state is a goal state, then stop and return success.
  - If it is better than the current state, then make it current state and proceed further.
  - If it is not better than the current state, then continue in the loop until a solution is found.

Step 5:Exit.

Advantages:

Hill climbing technique is useful in job shop scheduling, automatic programming, circuit designing, and vehicle routing and portfolio management. It is also helpful to solve pure optimization problems where the objective is to find the best state according to the objective function. It requires much less conditions than other search techniques.

Disadvantages:

The algorithm doesn't maintain a search tree, so the current node data structure need only record the state and its objective function value. It assumes that local improvement will lead to global improvement. There are some reasons by which hill climbing often gets suck which are stated below.

Local Maxima:

A local maxima is a state that is better than each of its neighbouring states, but not better than some other states further away. Generally this state is lower than the global maximum. At this point, one cannot decide easily to move in which direction! This difficulties can be extracted by the process of backtracking i.e. backtrack to any of one earlier node position and try to go on a different event direction.

To implement this strategy, maintaining in a list of path almost taken and go back to one of them. If the path was taken that leads to a dead end, then go back to one of them.

Figure Local Maxima

It is a special type of local maxima. It is a simply an area of search space. Ridges result in a sequence of local maxima that is very difficult to implement ridge itself has a slope which is difficult to traverse. In this type of situation apply two or more rules before doing the test. This will correspond to move in several directions at once.

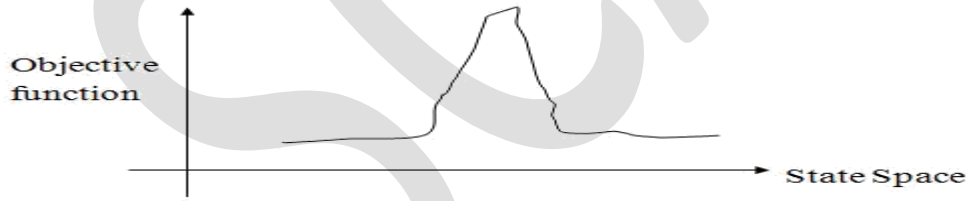


Figure Ridges

Plateau:

It is a flat area of search space in which the neighbouring have same value. So it is very difficult to calculate the best direction. So to get out of this situation, make a big jump in any direction, which will help to move in a new direction this is the best way to handle the problem like plateau.

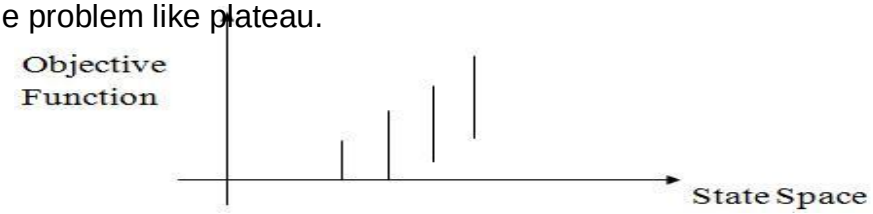


Figure Plateau

To overcome these problems we can

- Back track to some earlier nodes and try a different direction. This is a good way of dealing with local maxim.
- Make a big jump an some direction to a new area in the search. This can be done by applying two more rules of the same rule several times, before testing. This is a good strategy is dealing with plate and ridges. Hill climbing becomes inefficient in large problem spaces, and when combinatorial explosion occurs. But it is a useful when combined with other methods.

Steepest Descent Hill Climbing

- This is a variation of simple hill climbing which considers all the moves from the current state and selects the best one as the next state.
- Also known as Gradient search

Algorithm

- 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.
- Loop until a solution is found or until a complete iteration produces no change to current state:
  - Let SUCC be a state such that any possible successor of the current state will be better than SUCC
  - For each operator that applies to the current state do:
    - Apply the operator and generate a new state
    - Evaluate the new state. If is 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.
  - If the SUCC is better than the current state, then set current state to SYCC,

7. Explain the various problem solving methods. (December 2011)

- A problem in AI is defined as a task to be completed.
- Most of the problems are complex and cannot be solved by direct techniques.
- They can be solved by search strategies.
- Every search process can be viewed as a traversal of directed graph in which each node represents a problem state.

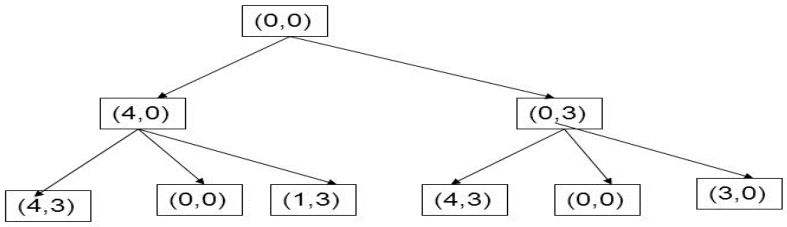
- Arcs represent the relationship between the states represented by the node it connects.
- The search process starts with an initial state and ends in a final state leading a path.
- Five important issues have to be considered.
    1. The direction to which to conduct the search.
    2. The topology of search process.
    3. How each node of the search process will be represented?
    4. Selecting applicable rules.
    5. Using a heuristic function to guide the search.
    - The various methods used to solve problems are
        1. Problem graph
        2. Matching
            a. indexing
        3. Heuristic functions.

1. Problem Graph

A simple way to implement the search strategy is tree traversal.

Each node of the tree is expanded by the production rules to generate a set of successor nodes, each of which can in turn be expanded, continuing until a node representing a solution is found.

Example: Water Jug Problem- two levels of Breadth first search Tree.



Implementing this procedure is

- Simple
- Require little bookkeeping.

Disadvantage

Same path may be generated many times and so processing takes place more than once.

This wastage can be avoided by bookkeeping and elimination of redundant nodes.

Graph search procedure

1. Examine the set of nodes that have been created so far to see if the new node already exists.
2. If it doesn't, add it to the graph just as for a tree.
3. If it is already present, then
    a. Set the node that is being expanded to point to already existing node corresponding to its successor, rather than new one. Throw the new node.
    b. If we keep track of the best path to each node, then check if the new path is best or worse.
        i. If worse do nothing.
        ii. If best record the new path as correct path and propagate the change in cost to all its successor nodes.

Uses

Graph search procedures are useful in partially commutative production system.

1. Matching

It is the process of applying a rule between the current state and precondition of the rules to determine some identity between them.

They are done by

1. Indexing
2. Matching with variables.
3. Complex matching
4. Filtering.

1. Indexing

- Use the current state as an index into rules and select the matching ones immediately.
- There are two problems
- It requires the use of large number of rules.
- It is unobvious that rules preconditions are satisfied by a particular state.
- Instead of using rules, use the current state as index and select the matching accordingly.
- In a game of chess, an index can be assigned to each board position. This scheme can be used while searching.

Disadvantage

- Indexing cannot be helpful when the preconditions of rules are stated as highly level predicates.

Advantage

- Indexing in some from is very efficient in rule based system.
- **Matching with variables**
- It is a simple kind of non literal match that require extensive search when patterns contain variables.
- **Complex and approximate matching.**

Rules should be applied if their preconditions approximately match the current situation

Example: A speech-understanding program

- **Filtering**

The result of matching is a list of rules whose left side have matched the current state description, along with variable binding that is generated by matching process.

- **Heuristic Function**
- A Heuristic function is a function that maps from problem state descriptions to measures of desirability usually represented as numbers.
- A Heuristic is a technique that improves the efficiency of a search process possibly by sacrificing claims of completeness.
- The purpose of heuristic function is to guide the search process in the most profitable directions suggesting which paths to follow first when more that one is available.

8. Explain Means- Ends Analysis with an example. (December 2011)

- Most of the search strategies either reason forward of backward however, often a mixture of the two directions is appropriate.
- Such mixed strategy would make it possible to solve the major parts of problem first and solve the smaller problems that arise when combining them together.
- Such a technique is called "Means - Ends Analysis".
- The means -ends analysis process centers around finding the difference between current state and goal state.

- The problem space of means - ends analysis has an initial state and one or more goal state, a set of operate with a set of preconditions their application and difference functions that computes the difference between two state a(i) and s(j).
- A problem is solved using means - ends analysis by

  1. Computing the current state s1 to a goal state s2 and computing their difference D12.

  2. Satisfy the preconditions for some recommended operator op is selected, then to reduce the difference D12.

  3. The operator OP is applied if possible. If not the current state is solved a goal is created and means- ends analysis is applied recursively to reduce the sub goal.

  4. If the sub goal is solved state is restored and work resumed on the original problem.(The first AI program to use means - ends analysis was the GPS General problem solver) Means- ends analysis is useful for many human planning activities.

  EXAMPLE

  - Problem for household robot: moving desk with 2 things on it from one room to another.
  - Main difference between start and goal state is location.
  - Choose PUSH and CARRY

Move desk with 2 things on it to new room

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

THE ROBOT OPERATORS

Operator	Preconditions	Results
PUSH (obj, loc)	at(robot,obj) &large (obj) &clear (obj) & arm empty	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)	Not holding (obj)
PLACE(obj1, obj2)	At(robot,obj2) & holding (obj1)	on(obj1, obj2)

A robot wants to move a desk with two things from one room to another. The object on top must be moved.

S B_____C G

Start PUSH Goal

If CARRY is chosen first, preconditions cannot be met. This results in two more differences that must be reduced. The location of the robot and the size of the desk.

If PUSH is chosen, then 3 preconditions has to be met. Two of which produce difference between the start and goal states.

The robot must be brought to current location be using WALK, the surface can be cleaned by two uses of PICKUP and PUTDOWN.

S(Start)_____**B**_____**C**

WALK(desk_loc)->PU(obj1)->PD(obj1)->PU(obj2)->PD(obj2) **PUSH**

Once PUSH is performed, the problem state is close to the goal state. The object must be placed back on the desk. PLACE can be used to put them there. But it cannot be applied immediately because the robot will be holding the objects.

S(Start)_____B_____ C E_____G(Goal)

WALK->PU(obj1)->PD(obj1)->PU(obj2)->PD(obj2) -> PUSH PLACE

Use WALK to get the robot back to the objects, followed by PICKUP and CARRY.

Final plan is

→ WALK (start_desk_loc) → PU(obj1) → PD(obj1) → PU(obj2)→
WALK(start_desk_loc) ← PUSH(desk, goal_loc) ← PD(obj2)
→ PU(obj1) → CARRY(obj1,goal_loc) → PLACE(obj1,desk) →
PLACE(obj2,desk) ← CARRY(obj2, goal_loc) ← PU(obj2) ← WALK(start)

Disadvantages

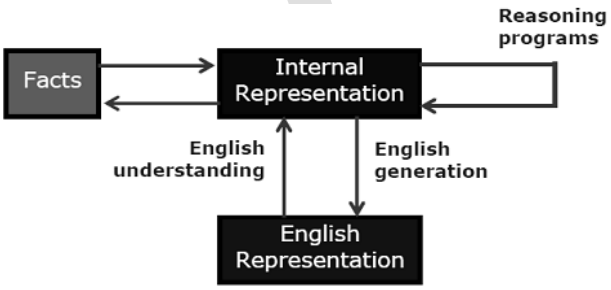
- It is not used to solve large and complex problems.
- The number of permutation of differences may get too large.
- Working on one difference may interfere with the plan for reducing another.
- Difference table would be immense for complex problems.

UNIT II REPRESENTATION OF KNOWLEDGE

Game playing - Knowledge representation, Knowledge representation using Predicate logic, Introduction to predicate calculus, Resolution, Use of predicate calculus, Knowledge representation using other logic-Structured representation of knowledge.

PART A

1. **Define Predicate Logic. (December 2011)**
   - Relational predicate logic is a generalization of propositional logic that is obtained by allowing proposition symbols to have arguments, and they are then called predicates.
   - The concepts and definitions for predicate logic are similar to those for propositional logic.
   - However, in order to establish the convention that all occurrences of a predicate have the same number of arguments one must start with the definition of a vocabulary.
2. **State the use of predicate calculus. (December 2011)**
   - This is used for capturing the essence of relationship between the classes.
   - We can represent complex facts using this calculus.
   - Resolving the rules is easy.
3. **Draw the mapping between facts and representations. (June 2013)**



4. **What is meant by alpha beta pruning? (MAY/JUNE 2016)**

The alpha beta cutoff strategy required the maintenance of two threshold values, one that represents a lower bound on the value that a maximizing node may have and another that represents a upper found on a value that minimizes node may be assigned.

5. **Write about iterative deepening. (November 2012, May 2014)**
   - It is a recursive procedure in which we start searching to a fixed depth in the game tree. We apply the single evaluation function to the first branch, applying to it the result of possible moves.
   - This will result in the initiation of a new minimax search, this time the depth getting increased to 2.
   - This process gets repeated.
6. **What is the difference between declarative knowledge and procedural knowledge? (May 2014)**

S.No	Declarative Knowledge	Procedural Knowledge
1	A declarative representation is one in which knowledge is specified, but the level of usage of that knowledge is unknown.	A procedural knowledge is one in which the control information that is needed to use the knowledge is assumed to be embedded within the knowledge itself.
2	Logical assertions are viewed as programs.	Logical assertions are viewed as data.
3	Reasoning paths define the viable execution paths	We need to augment viable the reasoning with an interpreter that proceeds according to instructions in the knowledge.
4	Control information exists as resides in constructs	Control information controls the knowledge base.

7. **Define conflict resolution.**

The result of matching process is a list of rules whose background have matched the current state description along with whatever variable bindings were generated by the matching process. It is the prerogative of the search method to decide the order in which the rules are to be applied, but sometimes we can incorporate the decision making in matching process. This phase is called conflict resolution.

8. **What is the purpose of unification? May-12,Dec-12**

It is used for finding substitutions for inference rules, which can make different logical expression to look identical. It helps to match to logical expressions. Therefore it is used in many algorithms in first order logic.

9. **What are the limitations in using propositional logic to represent knowledge base? May-11**

1) It has limited expressive power
2) It cannot directly represent properties of individuals or relations between individuals.
3) Generalizations, patterns, regularities cannot easily be represented.
4) Many rules are required to write so as to allow inference.

10. **Differentiate between propositional versus first order predicate logic. Dec-11**

Propositional logic	First order logic
Propositional logic is less expensive and do not reflect individual object's properties explicitly	First order logic is more expressive and can represent individual object along with all its properties.
Propositional logic cannot represent relationship among objects.	First order logic can represent relationship.
It does not consider generalization of objects.	It handles generalization
It includes sentence letters and logical connectives but not quantifier.	It has the same connectives as propositional logic, but it also has variables for individual objects, quantifier, symbols for functions and symbols for relations.

11. **What is unification algorithm?**

In propositional logic it is easy to determine that two literals cannot both be true at the same time. Simply look for L and ~L . In predicate logic, this matching process is more complicated, since bindings of variables must be considered.

For example man (john) and man(john) is a contradiction while man (john) and man(Himalayas) is not. Thus in order to determine contradictions we need a matching procedure that compares two literals and discovers whether there exist a set of substitutions that makes them identical. There is a recursive procedure that does this matching. It is called Unification algorithm.

In Unification algorithm each literal is represented as a list, where first element is the name of a predicate and the remaining elements are arguments. The argument may be a single element(atom) or may be another list

12. **For the Given sentence "All Pompeians were romans" write a well-formed formula in predicate logic. (MAY/JUNE 2016)**

    ∀X: Pompiens (x) → roman(x)

13. **How can you represent the resolution in predicate logic?**

    1. Convert all the statements of S to clausal form.
    2. Negate P and convert the result to clausal form. Add it to the set of clauses obtained in I.
    3. Repeat until a contradiction is found.

14. **List the canonical forms of resolution.**

    - eliminate implications
    - move negations down to the atomic formulas
    - purge existential quantifiers
    - rename variables if necessary
    - move the universal quantifiers to the left
    - move the disjunctions down to the literals
    - eliminate the conjunctions
    - rename the variables if necessary
    - purge the universal quantifiers

15. State the use of unification

- Syntactical first-order unification is used in logic programming and programming language type system implementation, especially in Hindley–Milner based type inference algorithms.
- Semantic unification is used in SMT solvers and term rewriting algorithms. Higher-order unification is used in proof assistants, for example Isabelle and Twelf, and restricted forms of higher-order unification (**higher-order pattern unification**) are used in some programming language implementations, such as lambda Prolog, as higher-order patterns are expressive.

PART B

1. i) State Representation of facts in predicate logic

ii) Explain resolution and unification algorithm in detail.(MAY/JUNE 2016)

Knowledge can be represented as *"symbol structures"* that characterize bits of knowledge about objects, concepts, facts, rules, strategies.

Examples: **"red"** represents *colour red*, **"car1"** represents *my car* , **"red(car1)"** represents fact that *my car is red*.

Assumptions about KR :

- *Intelligent Behavior* can be achieved by manipulation of symbol structures.
- *KR languages* are designed to facilitate operations over symbol structures, have precise syntax and semantics; *Syntax* tells which expression is legal?
  e.g., red1 (car1), red1 car1, car1(red1), red1(car1 & car2) ?; and *Semantic* tells what an expression means ?
  e.g., property "dark red" applies to my car.
- *Make Inferences, draw new conclusions* from existing facts.

To satisfy these assumptions about KR, we need formal notation that allows automated inference and problem solving. One popular choice is use of **logic**.

Logic

Logic is concerned with the truth of statements about the world. Generally each statement is either *TRUE* or *FALSE*. Logic includes: *Syntax, Semantics and Inference Procedure*.

1. Syntax:

Specifies the *symbols* in the language about how they can be combined to form sentences. The facts about the world are represented as sentences in logic.

2. Semantic:

Specifies how to assign a truth value to a sentence based on its *meaning* in the world. It Specifies what facts a sentence refers to. A fact is a claim about the world, and it may be *TRUE* or *FALSE*.

3. Inference Procedure:

Specifies *methods* for computing new sentences from the existing sentences.

Logic as a KR Language

Logic is a language for reasoning, a collection of rules used while doing logical reasoning. Logic is studied as KR languages in artificial intelligence. Logic is a formal system in which the formulas or sentences have true or false values.

Problem of designing KR language is a tradeoff between that which is

(a) Expressive enough to represent important objects and relations in a problem domain.

(b) Efficient enough in reasoning and answering questions about implicit information in a reasonable amount of time.

Logics are of different types: Propositional logic, Predicate logic, temporal logic, Modal logic, Description logic etc;

They represent things and allow more or less efficient inference.

Propositional logic and Predicate logic are fundamental to all logic.

Propositional Logic is the study of statements and their connectivity.

Predicate Logic is the study of individuals and their properties.

Logic Representation

Logic can be used to represent simple facts.

The *facts* are claims about the world that are *True* or *False*.

To build a Logic-based representation:

1. User defines a set of primitive *symbols* and the associated *semantics.*

2. Logic defines ways of putting symbols together so that user can define legal *sentences* in the language that represent *TRUE* facts.

3. Logic defines ways of inferring *new sentences* from existing ones.

4. Sentences - either *TRUE* or *false* but not both are called *propositions.*

5. A declarative sentence expresses a *statement* with a proposition as content; example:

the declarative **"snow is white"** expresses that **snow is white**; further, **"snow is white"** expresses that *snow is white* is **TRUE**.

Resolution and Unification algorithm

In propositional logic it is easy to determine that two literals cannot both be true at the same time. Simply look for L and ~L . In predicate logic, this matching process is more complicated, since bindings of variables must be considered.

For example man (john) and man(john) is a contradiction while man (john) and man(Himalayas) is not. Thus in order to determine contradictions we need a matching procedure that compares two literals and discovers whether there exist a set of substitutions that makes them identical. There is a recursive procedure that does this matching . It is called Unification algorithm.

In Unification algorithm each literal is represented as a list, where first element is the name of a predicate and the remaining elements are arguments. The argument may be a single element (atom) or may be another list. For example we can have literals as

(tryassassinate Marcus Caesar)

(tryassassinate Marcus (ruler of Rome))

To unify two literals, first check if their first elements re same. If so proceed. Otherwise they cannot be unified. For example the literals

(try assassinate Marcus Caesar)

(hate Marcus Caesar)

Cannot be unified. The unification algorithm recursively matches pairs of elements, one pair at a time. The matching rules are :

i) Different constants, functions or predicates can not match, whereas identical ones can.

ii) A variable can match another variable, any constant or a function or predicate expression, subject to the condition that the function or [predicate expression must not contain any instance of the variable being matched (otherwise it will lead to infinite recursion).

iii) The substitution must be consistent. Substituting y for x now and then z for x later is inconsistent. (a substitution y for x written as y/x)

The Unification algorithm is listed below as a procedure UNIFY (L1, L2). It returns a list representing the composition of the substitutions that were performed during the match. An empty list NIL indicates that a match was found without any substitutions. If the list contains a single value F, it indicates that the unification procedure failed.

UNIFY (L1, L2)

1. if L1 or L2 is an atom part of same thing do

(a) if L1 or L2 are identical then return NIL

(b) else if L1 is a variable then do

(i) if L1 occurs in L2 then return F else return (L2/L1)

© else if L2 is a variable then do

(i) if L2 occurs in L1 then return F else return (L1/L2)

else return F.

2. If length (L!) is not equal to length (L2) then return F.

3. Set SUBST to NIL

(at the end of this procedure , SUBST will contain all the substitutions used to unify L1 and L2).

4. For I = 1 to number of elements in L1 do

i) call UNIFY with the i th element of L1 and I'th element of L2, putting the result in S

ii) if S = F then return F

iii) if S is not equal to NIL then do

(A) apply S to the remainder of both L1 and L2

(B) SUBST := APPEND (S, SUBST) return SUBST.

Resolution yields a complete inference algorithm when coupled with any complete search algorithm. Resolution makes use of the inference rules. Resolution performs deductive inference. Resolution uses proof by contradiction. One can perform Resolution from a Knowledge Base. A Knowledge Base is a collection of facts or one can even call it a database with all facts.

Resolution basically works by using the principle of proof by contradiction. To find the conclusion we should negate the conclusion. Then the resolution rule is applied to the resulting clauses.

Each clause that contains complementary literals is resolved to produce a 2new clause, which can be added to the set of facts (if it is not already present). This process continues until one of the two things happen. There are no new clauses that

can be added. An application of the resolution rule derives the empty clause An empty clause shows that the negation of the conclusion is a complete contradiction, hence the negation of the conclusion is invalid or false or the assertion is completely valid or true.

Steps for Resolution

1. Convert the given statements in Predicate/Propositional Logic
2. Convert these statements into Conjunctive Normal Form
3. Negate the Conclusion (Proof by Contradiction)
4. Resolve using a Resolution Tree (Unification)

Steps to Convert to CNF (Conjunctive Normal Form)

1. Every sentence in Propositional Logic is logically equivalent to a conjunction of disjunctions of literals.

A sentence expressed as a conjunction of disjunctions of literals is said to be in Conjunctive normal Form or CNF.

1. Eliminate implication '→'

2. a → b = ~a v b

3. ~ (a ^ b) = ~ a v ~ b ……DeMorgan'sLaw

4. ~ (a v b) = ~ a ^ ~ b …….. DeMorgan'sLaw

5. ~ (~a) = a

Eliminate Existential Quantifier '∃'

To eliminate an independent Existential Quantifier, replace the variable by a Skolemconstant. This process is called as Skolemization.

Example: ∃y: President (y)

Here 'y' is an independent quantifier so we can replace 'y' by any name (say – George Bush).

So, ∃y: President (y) becomes President (George Bush).

To eliminate a dependent Existential Quantifier we replace its variable by SkolemFunction that accepts the value of 'x' and returns the corresponding value of 'y.'

Example: ∀x : ∃y : father_of(x, y)

Here 'y' is dependent on 'x', so we replace 'y' by S(x).

So, ∀x : ∃y : father_of(x, y) becomes ∀x : ∃y : father_of(x, S(x)).

Eliminate Universal Quantifier '∀'

To eliminate the Universal Quantifier, drop the prefix in PRENEX NORMAL FORM i.e. just drop ∀and the sentence then becomes in PRENEX NORMAL FORM.

Eliminate AND '^'

a ^ b splits the entire clause into two separate clauses i.e. a and b

(a v b) ^ c splits the entire clause into two separate clauses a v b and c

(a ^ b) v c splits the clause into two clauses i.e. a v c and b v c

To eliminate '^' break the clause into two, if you cannot break the clause, distribute the OR 'v' and then break the clause.

Problem Statement:

1. Ravi likes all kind of food.
2. Apples and chicken are food
3. Anything anyone eats and is not killed is food
4. Ajay eats peanuts and is still alive
5. Rita eats everything that Ajay eats

Prove by resolution that Ravi likes peanuts using resolution.

Step 1: Converting the given statements into Predicate/Propositional Logic

i. ∀x : food(x) → likes (Ravi, x)

ii. food (Apple) ^ food (chicken)

iii. ∀a : ∀b: eats (a, b) ^ ~killed (a) → food (b)

iv. eats (Ajay, Peanuts) ^ alive (Ajay)

v. ∀c : eats (Ajay, c) → eats (Rita, c)

vi. ∀d : alive(d) → ~killed (d)

vii. ∀e: ~killed(e) → alive(e)

Conclusion: likes (Ravi, Peanuts)

Step 2: Convert into CNF

i. ~food(x) v likes (Ravi, x)

ii. Food (apple)

iii. Food (chicken)

iv. ~ eats (a, b) v killed (a) v food (b)

v. Eats (Ajay, Peanuts)

vi. Alive (Ajay)

vii. ~eats (Ajay, c) V eats (Rita, c)

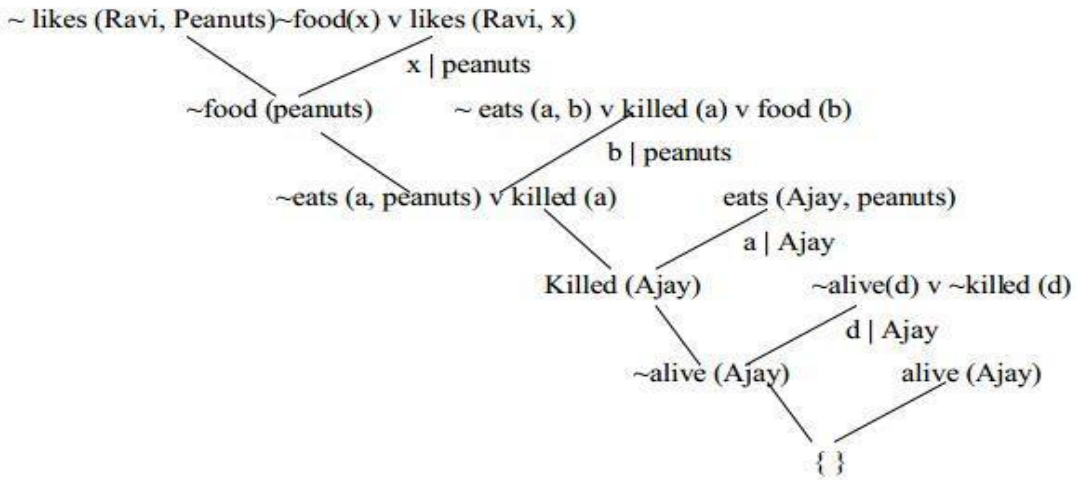
viii. ~alive (d) v ~ killed (d)

ix. Killed (e) v alive (e)

Conclusion: likes (Ravi, Peanuts)

Negate the conclusion

~ likes (Ravi, Peanuts)



Uses of Resolution in Today's World

- Used widely in AI.
- Helps in the development of computer programs to automate reasoning and theorem proving

2. Explain the various Issues in knowledge representation.

The fundamental goal of Knowledge Representation is to facilitate inference (conclusions) from knowledge.

The issues that arise while using KR techniques are many. Some of these are

1. Important Attributes: Any attribute of objects so basic that they occur in almost every problem domain?
2. Relationship among attributes: Any important relationship that exists among object attributes?
3. Choosing Granularity : At what level of detail should the knowledge be represented ?
4. Set of objects : How sets of objects be represented ?
5. Finding Right structure : Given a large amount of knowledge stored, how can relevant parts be accessed ?

1. Important Attributes:

There are attributes that are of general significance.

There are two attributes **"instance"** and **"isa"**, that are of general importance. These attributes are important because they support *property inheritance*.

2. Relationship among Attributes *:* The attributes to describe objects are themselves entities they represent.

The relationship between the attributes of an object, independent of specific knowledge they encode, may hold properties like:

Inverses, existence in an isa hierarchy, techniques for reasoning about values and single valued attributes.

1. Inverses : This is about *consistency check*, while a value is added to one attribute.The entities are related to each other in many different ways.

There are two ways of realizing this:

a. first, represent two relationships in a *single representation*; e.g., a logical representation, ***team(Pee-Wee-Reese, Brooklyn–Dodgers)***, that can be interpreted as a statement about Pee-Wee-Reese or Brooklyn–Dodger.

b. second, use attributes that focus on a *single entity but use them in pairs*, one the inverse of the other; for e.g., one, ***team = Brooklyn– Dodgers*** , and the other, ***team = Pee-Wee-Reese, ...***

This second approach is followed in semantic net and frame-based systems, accompanied by a knowledge acquisition tool that guarantees the consistency of inverse slot by checking, each time a value is added to one attribute then the corresponding value is added to the inverse.

2. Existence in an "isa" hierarchy :

This is about *generalization-specialization*, like, classes of objects and specialized subsets of those classes. There are attributes and specialization of attributes.

Example: the attribute *"height*" is a specialization of general attribute *"physical-size"* which is, in turn, a specialization of *"physical-attribute*".

These generalization-specialization relationships for attributes are important because they support inheritance.

3. Techniques for reasoning about values :

This is about *reasoning values of attributes* not given explicitly. Several kinds of information are used in reasoning, like, height : must be in a unit of length, age : of person can not be greater than the age of person's parents.

The values are often specified when a knowledge base is created.

4. Single valued attributes :

This is about a *specific attribute* that is guaranteed to take a unique value.

Example : A baseball player can at time have only a single height and be a member of only one team. KR systems take different approaches to provide support for single valued attributes.

3. Choosing Granularity

- What level should the knowledge be represented and what are the primitives ?
- Should there be a small number or should there be a large number of low-level primitives or High-level facts.
- High-level facts may not be adequate for inference while Low-level primitives may require a lot of storage.

Example of Granularity :

John spotted Sue. This could be represented as **Spotted (agent(John), object (Sue))**

4. Set of Objects

Certain properties of objects that is true as member of a set but not as individual.

This is done in different ways :

1. In logical representation through the use of *universal quantifier*, and

2. in hierarchical structure where node represent sets, the *inheritance propagate* set level assertion down to individual.

5. Finding Right Structure

- Access to right structure for describing a particular situation.
- It requires, selecting an initial structure and then revising the choice.
- While doing so, it is necessary to solve following problems :
    1. How to perform an initial selection of the most appropriate structure.
    2. How to fill in appropriate details from the current situations.
    3. How to find a better structure if the one chosen initially turns out not to be appropriate.
    4. What to do if none of the available structures is appropriate.
    5. When to create and remember a new structure.

3. Explain in detail about min-max procedure and alpha beta cutoffs. (May 2012, November 2012, May 2014)

ALPHA-BETA pruning is a method that reduces the number of nodes explored in Minimax strategy. It reduces the time required for the search and it must be restricted so that no time is to be wasted searching moves that are obviously bad for the current player.

The exact implementation of alpha-beta keeps track of the best move for each side as it moves throughout the tree.

We proceed in the same (preorder) way as for the minimax algorithm. For the MIN nodes, the score computed starts with +infinity and decreases with time.

For MAX nodes, scores computed starts with –infinity and increase with time.

The efficiency of the Alpha-Beta procedure depends on the order in which successors of a node are examined. If we were lucky, at a MIN node we would always consider the nodes in order from low to high score and at a MAX node the nodes in order from high to low score. In general it can be shown that in the most favorable circumstances the alpha-beta search opens as many leaves as minimax on a game tree with double its depth.

Alpha-Beta algorithm: The algorithm maintains two values, alpha and beta, which represents the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity. As the recursion progresses the "window" becomes smaller. When beta becomes less than alpha, it means that the current position cannot be the result of best play by both players and hence need not be explored further.

Pseudocode for the alpha-beta algorithm.

```
evaluate (node, alpha, beta)
if node is a leaf
return the heuristic value of node
if node is a minimizing node
for each child of node
beta = min (beta, evaluate (child, alpha, beta))
if beta <= alpha
return beta
```

return beta
if node is a maximizing node
for each child of node
alpha = max (alpha, evaluate (child, alpha, beta))
if beta <= alpha
return alpha

Min Max Algorithm

The Min-Max algorithm is applied in two player games, such as tic-tac-toe, checkers, chess, go, and so on.

There are two players involved, MAX and MIN. A search tree is generated, depth-first, starting with the current game position upto the end game position. Then, the final game position is evaluated from MAX's point of view, as shown in Figure 1. Afterwards, the inner node values of the tree are filled bottom-up with the evaluated values. The nodes that belong to the MAX player receive the maximum value of it's children. The nodes for the MIN player will select the minimun value of it's children.

The values represent how good a game move is. So the MAX player will try to select the move with highest value in the end. But the MIN player also has something to say about it and he will try to select the moves that are better to him, thus minimizing MAX's outcome.

Algorithm

```
MinMax (GamePosition game) {
return MaxMove (game);
}
MaxMove (GamePosition game)
 {
if (GameEnded(game))
 {
return EvalGameState(game);
}
else
{
best_move <- {};
moves <- GenerateMoves(game);
```

```
ForEach moves
{
move <- MinMove(ApplyMove(game));
if (Value(move) > Value(best_move))
{
best_move <- move;
}
}
return best_move;
}
}
MinMove (GamePosition game) {
best_move <- {};
moves <- GenerateMoves(game);
ForEach moves {
move <- MaxMove(ApplyMove(game));
if (Value(move) > Value(best_move)) {
best_move <- move;
}
}
return best_move;
}
```

Optimization

1. price
2. Limit the depth of the tree.

Speed up the algorithm

This all means that sometimes the search can be aborted because we find out that the search subtree won't lead us to any viable answer. This optimization is known as alpha-beta cutoffs.

The algorithm Have two values passed around the tree nodes:

The alpha value which holds the best MAX value found;

The beta value which holds the best MIN value found.

At MAX level, before evaluating each child path, compare the returned value with of the previous path with the beta value. If the value is greater than it abort the search for the current node;

At MIN level, before evaluating each child path, compare the returned value with of the previous path with the alpha value. If the value is lesser than it abort the search for the current node.

4. i) Write an algorithm for converting to clause form. (June 2013)

ii) Convert the following well – formed formula into clause form with sequence of steps,

∀ X: [Roman (x) ∩ Know (x, Marcus)] → [hate (x, caeser) ∪ (∀ y : ∈z: hate(y,z) →

Thinkcrazy (x,y))] (MAY/JUNE 2016)

To convert the axioms into conjunctive normal form (clause form)

1. Eliminate implications
2. Move negations down to the atomic formulas
3. Purge existential quantifiers
4. Rename variables if necessary
5. Move the universal quantifiers to the left
6. Move the disjunctions down to the literals
7. Eliminate the conjunctions
8. Rename the variables if necessary
9. Purge the universal quantifiers

Example

Consider the sentence

“All music lovers who enjoy Bach either dislike Wagner or think that anyone who dislikes any composer is a philistine".

Use enjoy() for enjoying a composer, and for dislike.

"x[musiclover(x)$\wedge$ enjoy(x,Bach)

dislike(x,Wagner) ("y[$z[dislike(y,z)] =>think-philistine(x,y)])]

Conversion

Step 1: Filter the expression to remove symbols.

"x [(musiclover(x) enjoy(x,Bach)) => dislike(x,Wagner)$\vee$ ("y[$z [dislike(y,z)]=> think-philistine(x,y)])].

Step 2: Filter using the following relationships:

¬(¬P) <=> P;

¬(a∧ b) <=> ¬a∨ ¬b;

¬(a∨ b) <=> ¬a∧ ¬b;

¬"x P(x) <=> $x ¬P(x); and

¬$x P(x) <=>"x ¬P(x).

Now the expression becomes

"x[¬musiclover(x)∨ ¬enjoy(x,Bach) ∨ dislike(x,Wagner) ∨"y["z[¬ dislike(y,z)] ∨think-philistine(x,y)]].

Step 3: Standardize variables so that each quantifier binds a unique variable.

"x Pred1(x)∨"x Pred2(x)

becomes

"x Pred1(x) ∨"y Pred2(y).

Step 4: Step 3 allows us to move all the quantifiers to the left in Step 4.

The expression becomes

"x"y"z[¬ musiclover(x) ∨¬enjoy(x,Bach) ∨

dislike(x,Wagner) ∨¬dislike(y,z)∨ think-philistine(x,y)].

This is called the prenex normal form.

Step 5: Eliminate existential quantifiers, by arguing that if $y Composer(y), then if we could actually find an Object S1 to replace the Variable x. So this gets replaced simply by Composer(S1).

Now, if existential quantifiers exist within the scope of universal quantifiers, we can't use merely an object, but rather a function that returns an object. The function will depend on the univeral quantifier.

"x$y tutor-of(y,x)

gets replaced by

"x tutor-of(S2(x),x).

This process is called Skolemization, and the S2 is a Skolem function.

Step 6: Any variable left must be universally quantified out on the left.

The expression becomes:

¬musiclover(x)∨ ¬enjoy(x,Bach) ∨ dislike(x,Wagner)∨ ¬dislike(y,z) ∨ think-philistine(x,y).

Step 7: Convert everything into a conjunction of disjunctions using the associative, commutative, distributive laws.

The form is

(a ∨ b ∨ c ∨ d ∨ ...) ∧ (p ∨ q ∨ ...) ∧ ...

Step 8: Call each conjunct a separate clause. In order for the entire wff to be true, each clause must be true separately.

Step 9: Standard apart the variables in the set of clauses generated in 7 and 8. This requires renaming the variables so that no two clauses make reference to the same variable. All variables are implicitly universally quantified to the left.

"x P(x) ∧ Q(x) <=> "xP(x) ∧ "xQ(x) <=> "xP(x) ∧ "yQ(y)

This completes the Procedure.

After application to a set of wffs, we end up with a set of clauses each of which is a disjunction of literals.

ii) Convert the following well – formed formula into clause form with sequence of steps,

∀ X: [Roman (x) ∩ Know (x, Marcus)] → [hate (x, caeser) ∪ (∀ y : ∈z: hate(y,z) →

Thinkcrazy (x,y))] (MAY/JUNE 2016)

Solution:

"All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy."

There are 9 simple steps to convert from Predicate logic to clause form.

We will use the following example :

" All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy."

The well-formed formula for this statement is :

"x [Roman(x) and know(x,Marcus)] ® [hate(x,Caesar) Ú ("y($z hate(y,z))® thinkcrazy(x,y))]

Well-formed formula [wff] : "x [Roman(x) and know(x,Marcus)] ®[hate(x,Caesar) Ú ("y($z hate(y,z)) ® think crazy(x,y))]

1. Eliminate

We will eliminate implication [®] by substituting it with its equivalent.

for e.g. a ® b = ~a v b .

Here 'a' and 'b' can be any predicate logic expression.

For the above statement we get :

"x ~[Roman(x) Ù know(x,Marcus)] Ú [hate(x,Caesar) Ú("y~($z hate(y,z)) Úthinkcrazy(x,y))]

2. Reduce the scope of ~

To reduce the scope we can use 3 rules :

~(~p) = p

DeMorgans Laws : ~(aÚb) = ~aÙ~b

~(aÙb) = ~aÚ~b

Applying this reduction on our example yields :

"x[~Roman(x) Ú~know (x,Marcus)] Ú [hate(x,Caesar) Ú ("y"z ~hate(y,z) Ú thinkcrazy(x,y))]

3.Change variable names such that, each quantifier has a unique name.

We do this in preparation for the next step. As variables are just dummy names, changing a variable name doesnot affect the truth value of the wff.

Suppose we have

"xP(x) Ú "xQ(x) will be converted to "x(P(x) Ú "yQ(y)

4.Move all the quantifiers to the left of the formula without changing their relative order.

As we already have unique names for each quantifier in the previous step, this will not cause a problem.

Performing this on our example we get :

"x"y"z [~Roman(x) Ú ~know(x,Marcus)] Ú [hate(x,Caesar)Ú(~hate(y,z)Úthinkcrazy(x,y))]

5. Eliminate existential quantifiers [$]

We can eliminate the existential quantifier by simply replacing the variable with a reference to a function that produces the desired value.

for eg. $y President(y) can be transformed into the formula President(S1)
If the existential quantifiers occur within the scope of a universal wuantifier, then the value that satisfies the predicate may depend on the values of the universally quantified variables.
For eg.. "x$y fatherof(y,x) will be converted to "x fatherof(S2(x),x)

6. Drop the Prefix

As we have eliminated all existential quantifiers, all the variables present in the wff are unversally quantified, hence for simplicity we can just drop the prefix, and assume that every variable is universally quantified.
We have form our example :
[~Roman(x) Ú ~know(x,Marcus)] Ú [hate(x,Caesar)Ú(~hate(y,z)Úthinkcrazy(x,y))]

7. Convert into conjunction of disjuncts

as we have no ANDs we will just have to use the associative property to get rid of the brackets.
In case of ANDs we will need to use the distributive property.
We have :
~Roman(x) Ú ~know(x,Marcus) Ú hate(x,Caesar) Ú ~hate(y,z) Ú thinkcrazy(x,y)

8. Separate each conjunct into a new clause.

As we did not have ANDs in out example, this step is avoided for our example and the final output of the conversion is :
~Roman(x) Ú ~know(x,Marcus) Ú hate(x,Caesar) Ú ~hate(y,z) Úthinkcrazy(x,y)

5. Explain the various approaches to knowledge representation. (June 2013)

A good knowledge representation enables fast and accurate access to knowledge and understanding of the content.
A knowledge representation system should have following properties.

1. Representational Adequacy: The ability to represent all kinds of knowledge that are needed in that domain.
2. Inferential Adequacy: The ability to manipulate the representational structures to derive new structure corresponding to new knowledge inferred from old.

3. Inferential Efficiency: The ability to incorporate additional information into the knowledge structure that can be used to focus the attention of the inference mechanisms in the most promising direction.
4. Acquisitional Efficiency: The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

Knowledge Representation Schemes

There are four types of Knowledge representation: Relational, Inheritable, Inferential, and Declarative/Procedural.

1. Relational Knowledge:

- Provides a framework to compare two objects based on equivalent attributes.
- Any instance in which two different objects are compared is a relational type of knowledge.

2. Inheritable Knowledge

- Is obtained from associated objects.
- It prescribes a structure in which new objects are created which may inherit all or a subset of attributes from existing objects.

3. Inferential Knowledge

- Is inferred from objects through relations among objects.
- e.g., a word alone is a simple syntax, but with the help of other words in phrase the reader may infer more from a word; this inference within linguistic is called semantics.

4. Declarative Knowledge

- A statement in which knowledge is specified, but the use to which that knowledge is to be put is not given.
- e.g. laws, people's name; these are facts which can stand alone, not dependent on other knowledge;

Procedural Knowledge

- A representation in which the control information, to use the knowledge, is embedded in the knowledge itself.
- e.g. computer programs, directions, and recipes; these indicate specific use or implementation.

UNIT III KNOWLEDGE INFERENCE

Knowledge representation -Production based system, Frame based system. Inference - Backward chaining, Forward chaining, Rule value approach, Fuzzy reasoning - Certainty factors, Bayesian Theory-Bayesian Network-Dempster - Shafer theory.

PART A

1. What is a production based system? (December 2011)

- A production based system is one
- That makes use of the production rules.
- Advantages of production rules:
  - They are modular,
  - Each rule defines a small and independent piece of knowledge.
  - New rules may be added and old ones deleted
  - Rules are usually independently of other rules.

2. State Bayesian Probability. (December 2011)

- Bayesian probability is one of the most popular interpretations of the concept of probability.
- The Bayesian interpretation of probability can be seen as an extension of logic that enables reasoning with uncertain statements.
- To evaluate the probability of a hypothesis, the Bayesian probability specifies some prior probability, which is then updated in the light of new relevant data.
- The Bayesian interpretation provides a standard set of procedures and formulae to perform this calculation.

3. What is non-monotonic inference? (May 2012)

Non monotonic inference is one that uses axioms and rules that are expanded so that complete information can be got from incomplete data. This system preserves the property that, at any instant of time, a statement is either believed to be true or false.

4. What is meant by rule based system. (November 2012)

- The rule based system is one that uses several rules as evidences and relates it to a single hypothesis.
- They use the hypothesis and evidences collected by various inferences and function.

5. When the knowledge is uncertain. How will you handle? (June 2013)

1. No monotonic logic
2. Probabilistic reasoning
3. Fuzzy logic
4. Truth values.

6. What is a Frame? (June 2013)

- A frame is a data structure with typical knowledge about a particular object or concept.
- Each frame has its own name and a set of **attributes** associated with it. *Name*, *weight*, *height* and *age* are slots in the frame *Person*.
- *Model*, *processor*, *memory* and *price* are slots in the frame *Computer*. Each attribute or slot has a value attached to it.
- Frames provide a natural way for the structured and concise representation of knowledge.

7. Define certainty factor. (November 2013)

A certainty factor CF(h,e) is defined in terms of two components.

- MB(h,e) is a measure of belief in hypothesis h given the evidence.
- MB measures the extent to which the evidence supports the hypothesis else it is zero it the evidence fails to support the hypothesis.
- MD(h,e) is the measure of disbelief in hypothesis h given the evidence e.
- MD measures the extent to which the evidence supports the negation else it is zero if the evidence fails to support the hypothesis.
- These two factors define certainty as CF(h,e)=MB(h,e)-MD(h,e)

8. When will you opt for a fuzzy system? (May 2014)

Fuzzy system is used when

1) The input information is noisy.
2) To solve complex problems.
3) Mathematical calculations are involved.

9. Define Baye's theorem. (May 2014)

A formula for determining conditional probability named after 18th-century British mathematician Thomas Bayes. The theorem provides a way to revise existing predictions or theories given new or additional evidence. In finance, Bayes' Theorem can be used to rate the risk of lending money to potential borrowers.

Formula:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A) * P(B|A)}{P(B)}$$

Also called "Bayes' Rule."

10. State the use of inference engine. (May 2014)

An **inference engine** interprets and evaluates the facts in the knowledge base in order to provide an answer. The **inference engine** enables the expert system to draw deductions from the rules in the KB

11. List down the applications of Bayesian network.

i) Turbo code generation.
ii) Build medical diagnosis system.
iii) Implementation of spam filters.

12. Define fuzzy logic and write the properties of fuzzy sets. (MAY/JUNE 2016).

FL is a problem-solving control system methodology that lends itself to implementation in systems ranging from simple, small, embedded micro-controllers to large, networked, multi-channel PC or workstation-based data

acquisition and control systems. It can be implemented in hardware, software, or a combination of both.

A fuzzy set A in X is characterized by membership function f_A (x) which associates with each point in X a real number in the interval [0,1], with the values of f_A (x) at x representing the grade of membership of x in A . Thus the nearer value of f_A (x) to unity the higher the grade of membership of x in A.

13. Define ontological engineering.

Ontology engineering in computer science and information science is a field which studies the methods and methodologies for building ontologies: formal representations of a set of concepts within a domain and the relationships between those concepts. A large-scale representation of abstract concepts such as actions, time, physical objects and beliefs would be an example of ontological engineering.

14. List the advantages of production rules.

Advantages of production rules:

- They are modular,
- Each rule defines a small and independent piece of knowledge.
- New rules may be added and old ones deleted
- Rules are usually independently of other rules.

15.Define Bayesian Network. (MAY/JUNE 2016)

A Bayesian network is directed acyclic graph whose nodes correspond to random variables; each node has a conditional distribution for the node, given its parents. Bayesian network provide a concise way to represent conditional independence relationships in the domain.

PART B

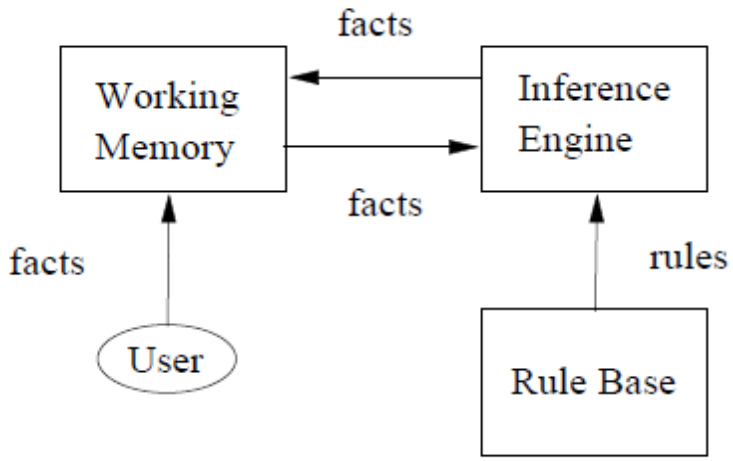
1. Explain about forward chaining and backward chaining with proper example. (June 2013, November 2013) (MAY/JUNE 2016)

FORWARD CHAINING

Forward chaining working from the facts to a conclusion. Sometimes called the data driven approach. To chain forward, match data in working memory against 'conditions‘ of rules in the rule-base. Starts with the facts, and sees what rules apply (and hence what should be done) given the facts.

WORKING

Facts are held in a working memory. Condition-action rules represent actions to take when specified facts occur in working memory. Typically the actions involve adding or deleting facts from working memory.



STEPS

•To chain forward, match data in working memory against 'conditions' of rules in the rule base.

•When one of them fires, this is liable to produce more data.

•So the cycle continues up to conclusion.

Example

•Here are two rules:

•If corn is grown on poor soil, then it will get blackfly.

•If soil hasn't enough nitrogen, then it is poor soil.

Forward chaining algorithm

Repeat

Collect the rule whose condition matches a fact in WM.

Do actions indicated by the rule.

(add facts to WM or delete facts from WM)

Until problem is solved or no condition match

Apply on the Example 2 extended (adding 2 more rules and 1 fact)

Rule R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : If fire THEN ADD switch_on_sprinklers

Rule R4 : IF dry THEN ADD switch_on_humidifier

Rule R5 : IF sprinklers_on THEN DELETE dry

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Fact F2 : Dry [Given]

Now, two rules can fire (R2 and R4)

Rule R4 ADD humidifier is on [from F2]

ADD smoky [from F1]

ADD fire [from F2 by R1]

ADD switch_on_sprinklers [by R3]

Rule R2

[followed by

sequence of

actions]

DELEATE dry, ie

humidifier is off **a conflict !** [by R5]

BACKWARD CHAINING

Backward chaining: working from the conclusion to the facts. Sometimes called the goal-driven approach. Starts with something to find out, and looks for rules that will help in answering it goal driven.

Steps in BC

•To chain backward, match a goal in working memory against 'conclusions' of rules in the rule-base.

•When one of them fires, this is liable to produce more goals.

•So the cycle continues

Example

•Same rules:

•If corn is grown on poor soil, then it will get blackfly.

•If soil hasn't enough nitrogen, then it is poor soil.

■ **Backward chaining algorithm**

Prove goal **G**

If **G** is in the initial facts , it is proven.

Otherwise, find a rule which can be used to conclude **G**, and try to prove each of that rule's conditions.

Encoding of rules

Rule R1 : IF hot AND smoky THEN fire

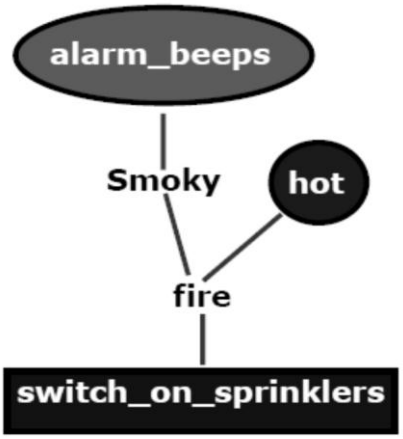
Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : If fire THEN switch_on_sprinklers

Fact F1 : hot [Given]

Fact F2 : alarm_beeps [Given]

Goal : Should I switch sprinklers on?



Example 1

Rule R1 : IF hot AND smoky THEN fire

Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : IF fire THEN switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Example 2

Rule R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : IF fire THEN ADD switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Example 3 : A typical Forward Chaining

Rule R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : If fire THEN ADD switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Fact F4 : smoky [from F1 by R2]

Fact F2 : fire [from F2, F4 by R1]

Fact F6 : switch_on_sprinklers [from F2 by R3]

Example 4 : A typical Backward Chaining

Rule R1 : IF hot AND smoky THEN fire

Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : If _fire THEN switch_on_sprinklers

Fact F1 : hot [Given]

Fact F2 : alarm_beeps [Given]

Goal : Should I switch sprinklers on?

2. Explain in detail about production and frame based system.

PRODUCTION BASED SYSTEM

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.

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

Situation ---------- Action

SITUATION that implies ACTION.

Example:-

IF the initial state is a goal state THEN quit.

The major components of an AI production system are

- A global database
- A set of production rules and

- A control system

The goal database is the central data structure used by an AI 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:-

- A monotonic production system
- A non-monotonic production system
- A partially commutative production system
- A commutative production system.

Advantages of production systems:-

- Production systems provide an excellent tool for structuring AI programs.
- Production Systems are highly modular because the individual rules can be added, removed or modified independently.
- 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.

Partially commutative, monotonic production systems are useful for solving ignorable problems. 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.

FRAME BASED SYSTEM.

A frame is a data structure with typical knowledge about a particular object or concept. Frames were first proposed by **Marvin Minsky** in the 1970s.

Each frame has its own name and a set of **attributes** associated with it. *Name*, *weight*, *height* and *age* are slots in the frame *Person*.

Model, *processor*, *memory* and *price* are slots in the frame *Computer*. Each attribute or slot has a value attached to it.

Frames provide a natural way for the structured and concise representation of knowledge.

A frame provides a means of organising knowledge in **slots** to describe various attributes and characteristics of the object.

Frames are an application of **object-oriented programming** for expert systems.

EXAMPLE

BOARDING PASS FRAME

QANTAS BOARDING PASS	
***Carrier*:**	*QANTAS AIRWAYS*
***Name*:**	*MR N BLACK*
***Flight*:**	*QF 612*
***Date*:**	*29DEC*
***Seat*:**	*23A*
***From*:**	*HOBART*
***To*:**	*MELBOURNE*
***Boarding*:**	*0620*
***Gate*:**	*2*

AIR NEW ZEALAND BOARDING PASS	
***Carrier*:**	*AIR NEW ZEALAND*
***Name*:**	*MRS J WHITE*
***Flight*:**	*NZ 0198*
***Date*:**	*23NOV*
***Seat*:**	*27K*
***From*:**	*MELBOURNE*
***To*:**	*CHRISTCHURCH*
***Boarding*:**	*1815*
***Gate*:**	*4*

Object-oriented programming is a programming method that uses *objects*as a basis for analysis, design and implementation.

In object-oriented programming, an **object** is defined as a concept, abstraction or thing with crisp boundaries and meaning for the problem at hand. All objects have identity and are clearly distinguishable. *Michael Black*, *Audi 5000 Turbo*, *IBM Aptiva S35* are examples of objects.

An object combines both data structure and its behaviour in a single entity. When an object is created in an object-oriented programming language, we first assign a name to the object, and then determine a set of attributes to describe the object's characteristics, and at last write procedures to specify the object's behaviour.

A knowledge engineer refers to an object as a **frame** (the term, which has become the AI jargon).*Frames as a knowledge representation technique*

The concept of a frame is defined by a collection of **slots**. Each slot describes a particular attribute or operation of the frame.

Slots are used to store values. A slot may contain a default value or a pointer to another frame, a set of rules or procedure by which the slot value is obtained.

Typical information included in a slot

Frame name.Relationship of the frame to the other frames.

The frame *IBM Aptiva S35* might be a member of the class *Computer*, which in turn might belong to the class *Hardware*.

Slot value.

A slot value can be **symbolic**, **numeric** or **Boolean**. For example, the slot *Name* has symbolic values, and the slot *Age* numeric values. Slot values can be assigned when the frame is created or during a session with the expert system.

Default slot value. The default value is taken to be true when no evidence to the contrary has been found. For example, a car frame might have four wheels and a chair frame four legs as default values in the corresponding slots.

Range of the slot value. The range of the slot value determines whether a particular object complies with the stereotype requirements defined by the frame. For example, the cost of a computer might be specified between $750 and $1500.

Procedural information. A slot can have a procedure attached to it, which is executed if the slot value is changed or needed.

Class and instances

The word *frame* often has a vague meaning. The frame may refer to a particular object, for example the computer *IBM Aptiva S35*, or to a group of similar objects. To be more precise, we will use the *instance-frame* when referring to a particular object, and the *class-frame* when referring to a group of similar objects.

A **class-frame** describes a group of objects with common attributes. *Animal*, *person*, *car* and *computer* are all class-frames.

Each frame "knows" its class.

EXAMPLE : COMPUTER CLASS

CLASS:	***Computer***		
[Str]	***Item Code*:**		
[Str]	***Model*:**		
[Str]	***Processor*:**		
[Str]	***Memory*:**		
[Str]	***Hard Drive*:**		
[Str]	***Floppy*:**	[Default]	*3.5"; 1.44MB*
[Str]	***CD-ROM*:**		
[Str]	***Mouse*:**		
[Str]	***Keyboard*:**		
[Str]	***Power Supply*:**	[Default]	*145 Watt*
[Str]	***Warranty*:**	[Default]	*3 years*
[N]	***Cost*:**		
[Str]	***Stock*:**	[Initial]	*In stock*

INSTANCES OF A CLASS

INSTANCE:	***IBM Aptiva S35***	
***Class*:**	*Computer*	
[Str]	***Item Code*:**	*SY7973*
[Str]	***Model*:**	*IBM Aptiva S35*
[Str]	***Processor*:**	*Pentium 233MHz*
[Str]	***Memory*:**	*48MB*
[Str]	***Hard Drive*:**	*6.4GB*
[Str]	***Floppy*:**	*3.5"; 1.44MB*
[Str]	***CD-ROM*:**	*24X*
[Str]	***Mouse*:**	*Cordless Mouse*
[Str]	***Keyboard*:**	*104-key*
[Str]	***Power Supply*:**	*145 Watt*
[Str]	***Warranty*:**	*3 years*
[N]	***Cost*:**	*1199.99*
[Str]	***Stock*:**	*In stock*

INSTANCE:	***IBM Aptiva S9C***	
***Class*:**	*Computer*	
[Str]	***Item Code*:**	*SY7975*
[Str]	***Model*:**	*IBM S9C*
[Str]	***Processor*:**	*Pentium 200MHz*
[Str]	***Memory*:**	*32MB*
[Str]	***Hard Drive*:**	*4.2GB*
[Str]	***Floppy*:**	*3.5"; 1.44MB*
[Str]	***CD-ROM*:**	*16X*
[Str]	***Mouse*:**	*2-button mouse*
[Str]	***Keyboard*:**	*104-key*
[Str]	***Power Supply*:**	*145 Watt*
[Str]	***Warranty*:**	*3 years*
[N]	***Cost*:**	*999.99*
[Str]	***Stock*:**	*In stock*

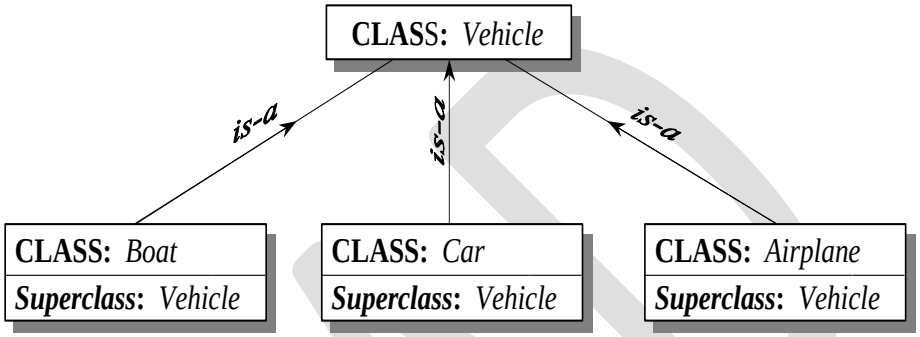
Class inheritance in frame-based systems

Frame-based systems support **class inheritance**.

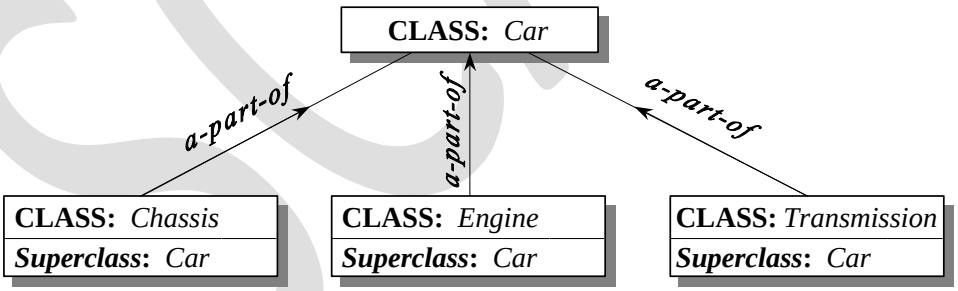
The fundamental idea of inheritance is that attributes of the class-frame represent things that are *typically* true for all objects in the class. However, slots in the instance-frames can be filled with actual data uniquely specified for each instance.

Relationships among objects

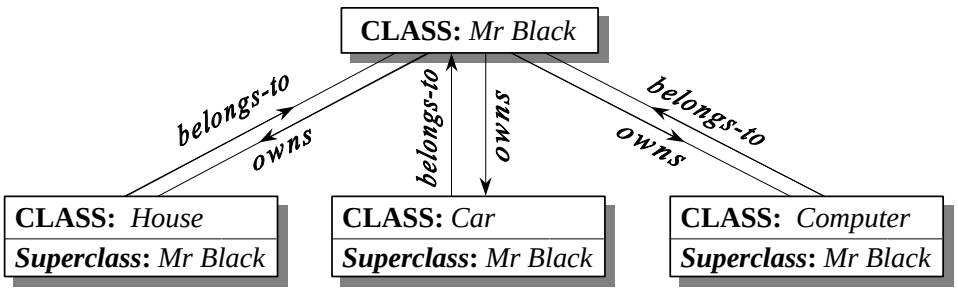
1. **Generalisation** denotes ***a-kind-of*** or ***is-a*** relationship between superclass and its subclasses. For example, a car *is a* vehicle, or in other words, *Car* represents a subclass of the more general superclass *Vehicle*. Each subclass inherits all features of the superclass



2. **Aggregation** is ***a-part-of*** or ***part-whole*** relationship in which several subclasses representing *components* are associated with a superclass representing a *whole*. For example, an engine is *a part of* a car.



3. **Association** describes some semantic relationship between different classes which are unrelated otherwise. For example, Mr Black owns a house, a car and a computer. Such classes as *House*, *Car* and *Computer* are mutually independent, but they are linked with the frame *Mr Black* through the semantic association.



Methods and demons

Expert systems are required not only to store the knowledge but also to validate and manipulate this knowledge. To add actions to our frames, we need **methods** and **demons**.

A method is a procedure associated with a frame attribute that is executed whenever requested.

We write a method for a specific attribute to determine the attribute's value or execute a series of actions when the attribute's value changes.

Most frame-based expert systems use two types of methods:

WHEN CHANGED and WHEN NEEDED.

3. Explain the need of fuzzy set and fuzzy logic with example. (May-12) / Briefly explain how reasoning is done using fuzzy logic. (MAY/JUNE 2016)

Fuzzy Set

- The word "fuzzy" means "vagueness". Fuzziness occurs when the boundary of a piece of information is not clear-cut.
- Fuzzy sets have been introduced by Lotfi A. Zadeh (1965) as an extension of the classical notion of set.
- Classical set theory allows the membership of the elements in the set in binary terms, a bivalent condition - an element either belongs or does not belong to the set.

➢ Fuzzy set theory permits the gradual assessment of the membership of elements in a set, described with the aid of a membership function valued in the real unit interval [0, 1].

Fuzzy Set Theory

Fuzzy set theory is an extension of classical set theory where elements have varying degrees of membership. A logic based on the two truth values, True and False, is sometimes inadequate when describing human reasoning. Fuzzy logic uses the whole interval between 0 (false) and **1** (true) to describe human reasoning.

➢ **Fuzzy logic** is derived from fuzzy set theory dealing with reasoning that is approximate rather than precisely deduced from classical predicate logic.

➢ Fuzzy logic is capable of handling inherently imprecise concepts.

➢ Fuzzy set theory defines Fuzzy Operators on Fuzzy Sets.

Crisp and Non-Crisp Set

- The **characteristic function** μ**A(x)** whichhas values **0** ('false') and **1** ('true") only are **crisp sets.**
- For Non-crisp sets the characteristic function μ**A(x)** can be defined.

  The characteristic function μ**A(x)** for the crisp set is generalized for the Non-crisp sets.
  - This generalized characteristic function μ**A(x)** is called **membership function**.
  - Such Non-crisp sets are called **Fuzzy Sets**.
  - Crisp set theory is not capable of representing descriptions and classifications in many cases, In fact, Crisp set does not provide adequate representation for most cases.

Representation of Crisp and Non-Crisp Set

Example:

Classify students for a basketball team

This example explains the grade of truth value.

- **tall students** qualify and **not tall students** do not qualify
- if students 1.8 m tall are to be qualified, then should we exclude a student who is 1/10" less? Or should we exclude a student who is 1" shorter?
- Non-Crisp Representation to represent the notion of a tall person

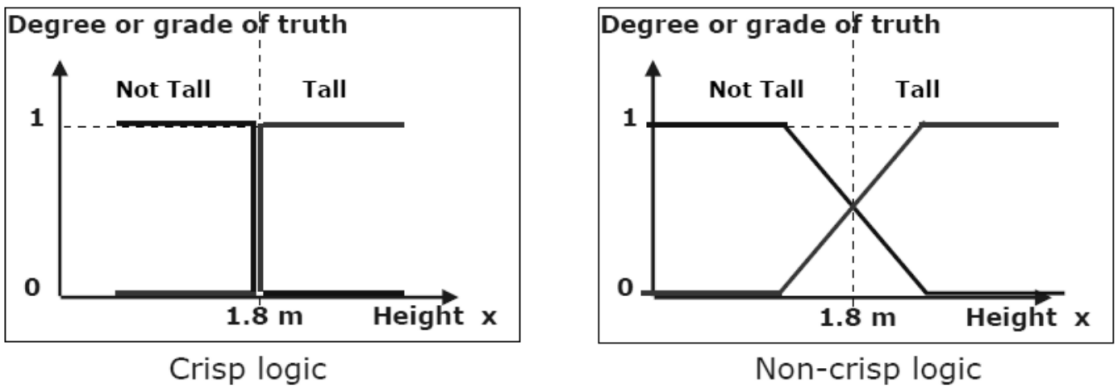


Fig. 1 Set Representation – Degree or grade of truth

- A student of height 1.79m would belong to both tall and not tall sets with a particular degree of membership.
- As the height increases the membership grade within the tall set would increase whilst the membership grade within the not-tall set would decrease.

Capturing Uncertainty

Instead of avoiding or ignoring uncertainty, Lotfi Zadeh introduced Fuzzy Set theory that captures uncertainty.

- A fuzzy set is described by a **membership function** μ**A (x)** of **A**. This membership function associates to each element **x**σ Є **X** a number as μ**A (x**σ **)** in the closed unit interval **[0, 1]**.
- The number μ**A (x**σ **)** represents the **degree of membership** of **x**σ in **A**.
- The notation used for membership function μ**A (x)** of a fuzzy set **A** is **A : X** Є **[0, 1]**
- Each membership function maps elements of a given universal base set **X** , which is itself a crisp set, into real numbers in **[0, 1] .**

Example:

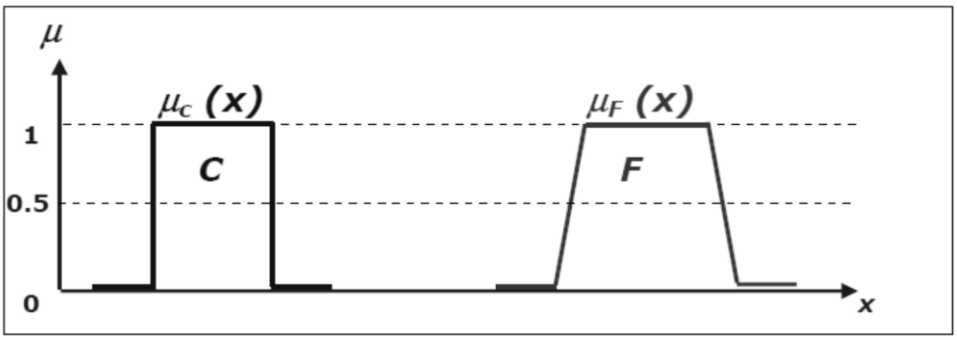


Fig. 2 Membership function of a Crisp set C and Fuzzy set F

- In the case of Crisp Sets the members of a set are : either out of the set, with membership of degree " **0** ", or in the set, with membership of degree " **1** ",
  - Therefore, **Crisp Sets ЄFuzzy Sets**

  In other words, Crisp Sets are Special cases of Fuzzy Sets.

Examples of Crisp and Non-Crisp Set

Example 1: Set of prime numbers (a crisp set)

- If we consider space **X** consisting of natural numbers ≤ **12** ie **X = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}**
- Then, the set of prime numbers could be described as follows.

PRIME = {x contained in X | x is a prime number} = {2, 3, 5, 6, 7, 11}

Fuzzy Set

A Fuzzy Set is any set that allows its members to have different degree of membership, called membership function, in the interval **[0 , 1]**.

Definition of Fuzzy set

A **fuzzy set A**, defined in the universal space **X**, is a function defined in **X** which assumes values in the range **[0, 1]**.

A fuzzy set **A** is written as a set of pairs **{x, A(x)}** as

A = {{x , A(x)}} , x in the set X

where **x** is an element of the universal space **X**, and

A(x) is the value of the function **A** for this element.

The value **A(x)** is the **membership grade** of the element **x** in a fuzzy set **A**.

Example : Set **SMALL** in set **X** consisting of natural numbers ≤ **to 12**.

Assume: SMALL(1) = 1, SMALL(2) = 1, SMALL(3) = 0.9, SMALL(4) = 0.6, SMALL(5) = 0.4, SMALL(6) = 0.3, SMALL(7) = 0.2, SMALL(8) = 0.1, SMALL(u) = 0 for u >= 9.

Then, following the notations described in the definition above :

Set SMALL = {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3}, {7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}

Note that a fuzzy set can be defined precisely by associating with each **x** , its grade of membership in **SMALL**.

• **Definition of Universal Space**

Originally the universal space for fuzzy sets in fuzzy logic was defined only on the integers. Now, the universal space for fuzzy sets and fuzzy relations is defined with three numbers.

The first two numbers specify the start and end of the universal space, and the third argument specifies the increment between elements.

This gives the user more flexibility in choosing the universal space.

Example : The fuzzy set of numbers, defined in the universal space **X = { xi } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}** is presented as **SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]**

Fuzzy Membership

- A fuzzy set **A** defined in the universal space **X** is a function defined in **X** which assumes values in the range **[0, 1]**.
- A fuzzy set **A** is written as a set of pairs **{x, A(x)}**.

  A = {{x , A(x)}} , x in the set X

  where **x** is an element of the universal space **X**, and

A(x) is the value of the function **A** for this element.

The value **A(x)** is the **degree of membership** of the element **x** in a fuzzy set **A**. The Graphic Interpretation of fuzzy membership for the fuzzy sets :

• Graphic Interpretation of Fuzzy Sets SMALL

The fuzzy set SMALL of small numbers, defined in the universal space **X = { xi } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}** is presented as

SetOption [Fuzzy Set, Universal Space → {1, 12, 1}]

The Set **SMALL** in set **X** is : **SMALL = FuzzySet {{1, 1 }, {2, 1 }, {3, 0.9}, {4, 0.6}, {5, 0.4}, {6, 0.3},{7, 0.2}, {8, 0.1}, {9, 0 }, {10, 0 }, {11, 0}, {12, 0}}**

Therefore **SetSmall** is represented as

SetSmall = **FuzzySet [{{1,1},{2,1}, {3,0.9}, {4,0.6}, {5,0.4},{6,0.3}, {7,0.2}, {8, 0.1}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}** , **UniversalSpace → {1, 12, 1}]**

Fuzzy Plot [SMALL, AxesLable → {"X", "SMALL"}]

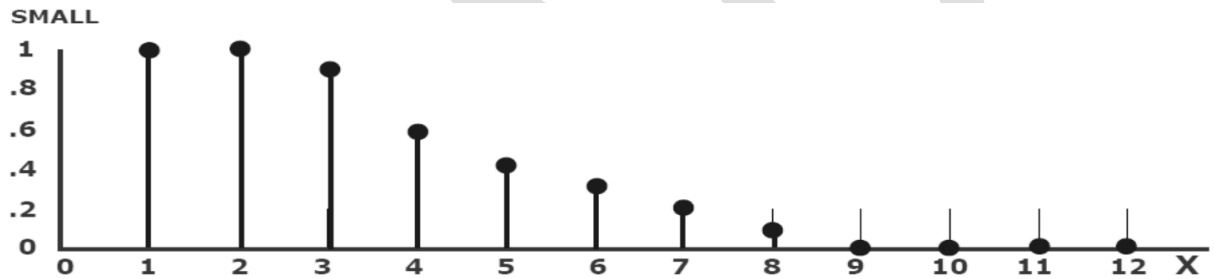


Fig Graphic Interpretation of Fuzzy Sets SMALL

Graphic Interpretation of Fuzzy Sets EMPTY

- An empty set is a set that contains only elements with a grade of membership equal to **0**.

  Example: Let EMPTY be a set of people, in Minnesota, older than 120.
- The Empty set is also called the Null set.
- The fuzzy set **EMPTY** , defined in the universal space **X = { xi } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}** is presented as **SetOption [FuzzySet, UniversalSpace → {1, 12, 1}]**
- The Set **EMPTY** in set **X** is : **EMPTY = FuzzySet {{1, 0}, {2, 0}, {3, 0}, {4, 0}, {5, 0}, {6, 0}, {7, 0},{8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}**
- Therefore **SetEmpty** is represented as **SetEmpty** = **FuzzySet [{{1,0},{2,0}, {3,0}, {4,0}, {5,0},{6,0}, {7,0},{8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0}}** , **UniversalSpace → {1, 12, 1}]**
- **FuzzyPlot [EMPTY, AxesLable ={"X", " UNIVERSAL SPACE "}]**

4. Explain how Bayesian network is used for representing knowledge in an uncertain domain.

- Bayesian networks (BN) are a network-based framework for representing and analyzing models involving uncertainty;
- BN are different from other knowledge-based systems tools because uncertainty is handled in mathematically rigorous yet efficient and simple way.
- BN are different from other probabilistic analysis tools because of network representation of problems, use of Bayesian statistics, and the synergy between these.

Definition of a Bayesian Network

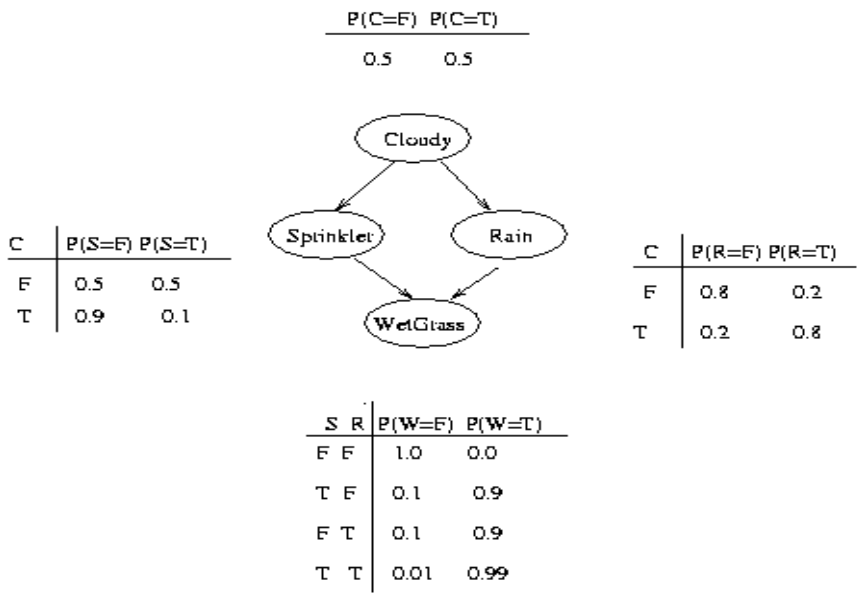
Knowledge structure:

- variables are nodes
- arcs represent probabilistic dependence between variables
- conditional probabilities encode the strength of the dependencies

Computational architecture:

- computes posterior probabilities given evidence about some nodes
- exploits probabilistic independence for efficient computation

- A Bayesian network is a representation of the joint distribution over all the variables represented by nodes in the graph. Let the variables be X(1), ..., X(n).
- Let parents(A) be the parents of the node A. Then the joint distribution for X(1) through X(n) is represented as the product of the probability distributions P(Xi|Parents(Xi)) for i = 1 to n. If X has no parents, its probability distribution is said to be unconditional, otherwise it is conditional.



By the chaining rule of probability, the joint probability of all the nodes in the graph above is:

P(C, S, R, W) = P(C) * P(S|C) * P(R|C) * P(W|S,R)

W=Wet Grass, C=Cloudy, R=Rain, S=Sprinkler

Example: P(W∩-R∩S∩C)

= P(W|S,-R)*P(-R|C)*P(S|C)*P(C)

= 0.9*0.2*0.1*0.5 = 0.009

What is the probability of wet grass on a given day - P(W)?

P(W) = P(W|SR) * P(S) * P(R) +

P(W|S-R) * P(S) * P(-R) +

P(W|-SR) * P(-S) * P(R) +

P(W|-S-R) * P(-S) * P(-R)

Here P(S) = P(S|C) * P(C) + P(S|-C) * P(-C)

P(R) = P(R|C) * P(C) + P(R|-C) * P(-C)

P(W)= 0.5985

Advantages of Bayesian Approach

- Bayesian networks can readily handle incomplete data sets.
- Bayesian networks allow one to learn about causal relationships
- Bayesian networks readily facilitate use of prior knowledge.

5. Explain in detail about Ruled based approach and Dempster –Shafer theory.(MAY/JUNE 2016)

Ruled based approach

Rule-Based system architecture consists *a set of rules*, *a set of facts*, and an *inference engine*.

Types of Rules

Three types of rules are mostly used in the Rule-based production systems.

- **Knowledge Declarative Rules :**

  These rules state all the facts and relationships about a problem.

  Example :

  IF inflation rate declines

  THEN the price of gold goes down.

  These rules are a part of the knowledge base.

- **Inference Procedural Rules**

  These rules advise on how to solve a problem, while certain facts are known.

  Example :

  IF the data needed is not in the system

  THEN request it from the user.

  These rules are part of the inference engine.

- **Meta rules**

  These are rules for making rules. Meta-rules reason about which rules should be considered for firing.

  Example :

  IF the rules which do not mention the current goal in their premise,

  AND there are rules which do mention the current goal in their premise,

  THEN the former rule should be used in preference to the latter.

  Meta-rules direct reasoning rather than actually performing reasoning.

  Meta-rules specify which rules should be considered and in which order they should be invoked. FACTS :They represent the real world information

Inference Engine

The inference engine uses one of several available forms of inferencing. By inferencing means the method used in a knowledge-based system to process the stored knowledge and supplied data to produce correct conclusions.

Example

How old are you?

Subtract the year you were born in from 2014.

The answer will either be exactly right,

Or one year short.

Dempster/Shafer theory

- The Dempster-Shafer theory, also known as the theory of belief functions, is a generalization of the Bayesian theory of subjective probability.
- The Bayesian theory requires probabilities for each question of interest, belief functions allow us to base degrees of belief for one question on probabilities for a related question.
- These degrees of belief may or may not have the mathematical properties of probabilities; how much they differ from probabilities will depend on how closely the two questions are related.
- The Dempster-Shafer theory owes its name to work by A. P. Dempster (1968) and Glenn Shafer (1976), but the kind of reasoning the theory uses can be found as far back as the seventeenth century.
- The theory came to the attention of AI researchers in the early 1980s, when they were trying to adapt probability theory to expert systems.
- Dempster-Shafer degrees of belief resemble the certainty factors in MYCIN, and this resemblance suggested that they might combine the rigor of probability theory with the flexibility of rule-based systems.
- Subsequent work has made clear that the management of uncertainty inherently requires more structure than is available in simple rule-based systems, but the Dempster-Shafer theory remains attractive because of its relative flexibility.
- The Dempster-Shafer theory is based on two ideas: the idea of obtaining degrees of belief for one question from subjective probabilities for a related question, and Dempster's rule for combining such degrees of belief when they are based on independent items of evidence.

UNIT IV PLANNING AND MACHINE LEARNING

Basic plan generation systems - Strips -Advanced plan generation systems – K strips -Strategic explanations -Why, Why not and how explanations. Learning-Machine learning, adaptive Learning.

PART A

1. What are the various planning systems? (November 2012)

Other planning techniques are

1. Triangle tables – provides a way to record the goals that each operator is expected to satisfy as well as the goals that must be true for it to execute correctly. If something unexpected happens during the execution of a plan, the table provides information required to patch the plan.
2. Meta planning – a technique for reasoning not just the problem being solved but also planning process itself.
3. Macro operators – allow a planner to build new operators that represent commonly used sequences of operators.
4. Case based planning – Reuse old plans to make new ones.

2. Define plan. (June 2013)

Plan is a way of decomposing a problem into some parts and the various techniques can be applied on it to handle the various interactions as they are detected during the problem solving process.

3. What do you mean by goal stack planning? (June 2013)

i. One of the earliest techniques is planning using goal stack.
ii. Problem solver uses single stack that contains
    1. sub goals and operators both
    2. Sub goals are solved linearly and then finally the conjoined sub goal is solved.
iii. Plans generated by this method will contain
    1. Complete sequence of operations for solving one goal followed by complete sequence of operations for the next etc.
iv. Problem solver also relies on
    1. A database that describes the current situation.
    2. Set of operators with precondition, add and delete lists.

4. Define machine learning. (June 2013, November 2013)

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data.

5. What is planning? (November 2013)

Planning refers to the use of methods that focus on ways of decomposing the original problem into appropriate subparts and on ways of recording and handling interaction among the subparts as they are detected during the problem solving process.

6. State the components of a planning system. (May 2014)

1. Choosing rules to apply.
2. Applying rules.
3. Detecting a solution.
4. Detecting dead ends.
5. Repairing an almost correct solution.

7. Define reactive system.

- A reactive system is one that has access to a knowledge base of some sort that describes the actions to be taken under different circumstances.
- It does not have the power of anticipation.

8. How is the complexity of learning measured?

The complexity of learning is a function of 3 factors

1. Error tolerance (h).
2. The number of binary features present in the example(t)
3. The size of the rule necessary to make the discrimination.(f).

9. What is explanation based learning? May-10

Explanation based learning is a method for extracting rules from individual observations through an explanation.General rule follows logically from the background of the Cavemen's usual cooking process.

10. List the various issues that affect the design of a learning element.

There are several factors affecting the performance. They are,

- Types of training provided.
- The form and extent of any initial background knowledge.
- The type of feedback provided.
- The learning algorithms used.

11. Brief frame problem. (MAY/JUNE 2016)

Frame problem defines an issue with using first order logic to express facts about a robot in the world. Representing the state of a robot with traditional FOL requires the use of many axioms that simply imply that things in the environment do not change arbitrarily

.

12. What is Q learning?

Q-learning uses temporal differences to estimate the value of Q*(s,a).In Q-learning, the agent maintains a table of Q[S,A], where S is the set of states and A is the set of actions. Q[s,a] represents its current estimate of Q*(s,a).

An experience s,a,r,s' provides one data point for the value of Q(s,a). The data point is that the agent received the future value of r+ γV(s'), where V(s') =maxa' Q(s',a'); this is the actual current reward plus the discounted estimated future value. This new data point is called a return. The agent can use the temporal difference equation to update its estimate for Q(s,a):

$$Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$$

or, equivalently,

$$Q[s,a] \leftarrow (1-\alpha)\, Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s',a']).$$

This assumes that α is fixed; if α is varying, there will be a different count for each state-action pair and the algorithm would also have to keep track of this count.

13. What is rote learning? (MAY/JNUE 2016)

Simple sorting of computed information or rote learning, is the most basic learning activity. Many Computer programs, e.g database management system, can be said to learn in this sense although most people would not call such simple storage learning.

14. Explain the concept of learning with example.

Learning involves

- **Task**

The behavior or task that is being improved

- **Data**

The experiences that are used to improve performance in the task

- **Measure of improvement**

How the improvement is measured - for example, new skills that were not present initially, increasing accuracy in prediction, or improved speed

15. Distinguish between supervised learning and unsupervised learning.

Dec-11, May- 13

Supervised learning involves learning a function from example of its input and outputs whereas unsupervised learning involves learning patterns in the input when no specific output values are supplied.

In supervised learning the environment is fully observable where as in unsupervised learning the environment is partially observable.

PART B

1. Briefly explain the advanced plan generation systems. (December 2011)/ Describe hierarchical planning methods with an example.(MAY/JUNE 2016)

Nonlinear Planning using Constraint Posting

- Idea of constraint posting is to build up a plan by incrementally
- Hypothesizing operators,
- Partial ordering between operators and
- Binding of variables within operators
- At any given time in planning process, a solution is a partially ordered, partially instantiated set of operators.
- To generate actual plan, convert the partial order into any of a number of total orders.

Main steps involved in non linear plan generation are:

- Step addition - Creating new operator (step) for a plan.
- Promotion - Constraining one operator to come before another in final plan

- Declobbering - Placing operator Op2 between two operators Op1 and Op3 such that Op2 reasserts some pre conditions of Op3 that was negated by Op1
- Simple Establishment Assigning a value to a variable, in order to ensure the pre conditions of some step.

Algorithm:

- Initialize S to be set of propositions in the goal state.
- Remove some unachieved proposition P from S.
- Achieve P by using step addition, promotion, declobbering, simple establishment.
- Review all the steps in the plan, including any new steps introduced by step addition to see if any of their preconditions are unachieved.
- Add to S the new set of unachieved preconditions.
- If S = φ, complete the plan by converting the partial order of steps into a total order and instantiate any variables as necessary and exit.
- Otherwise go to step 2.

Hierarchical planning

- Main problem in strips-like planning (as well as in other planning frameworks): complexity.
- One reason for complexity: no structure
- No distinction between important and unimportant properties
- No distinction between important and unimportant operators.
- This observation gives raise to two different ways of abstraction in planning:
    i. Abstraction of situations and
    ii. Abstraction of operators.

Abstraction of Situations - abstrips (Abstraction-Based strips)

Main idea: introduce weights for each literal and consider only the most important ones in first loop, then refine by considering also literals with second highest weight.

In blocksworld, for instance, properties with weights:

Property	Weight
On	4
Clear	3
Holds	2
Ontable	2
Handempty	1

Higher weights indicate more important properties. Means here concretely: first consider only the property On, in the second loop the properties On and Clear and so on. Use the abstract plan for a refinement of the more detailed plans. In the last loop all properties have to be considered.

Algorithm

Assume non-linear planning algorithm NLP with input: partial nonlinear plan & output: total well-formed conflict-free non-linear plan.NLPAS (Non-Linear Planning with Abstraction of Situations) calls NLP initially considering only the most important properties. This plan is refined by taking into account less important properties.

Input: non-linear plan and index (initially the highest weight)

begin

if Index=0 then return(P)

else

disregard all preconditions of operators,

whose weights are smaller than Index

select P':=NLP(P) ;;; arbitrary choice → backtrack point

endif

return(NLPAS(P',Index-1)) end

Problems with abstraction

Abstraction certainly plays an important role in human problem solving. Selection of useful order is not possible.

Also maintenance of visibility is not done

Abstraction of Operators

Operators of type PICKUP is divided into three parts:

- POSITION for positioning the robot's hand.
- CATCH for grasping it.
- LIFT to lift it.

Pseudocode

```
input: initial non-linear plan output: non-linear plan
begin
while nodes unsolved or unexpanded in P do
select unsolved or unexpanded node N from P
arbitrary choice → backtrack point
if N unsolved
then
select action A, which has N
in add list ;;; arbitrary choice → backtrack point
insert A as operator node immediately
before N in P add achieve-nodes for the preconditions of A
in parallel immediately before A
else
replace N by its plot and supplement preconditions
end if
resolve interactions
criticise plan
end do
return(P)  end
```

2. **Explain the components of a planning system with suitable examples.(November 2012) (MAY/JUNE 2016)**

Planning refers to the use of methods that focus on ways of decomposing the original problem into appropriate subparts and on ways of recording and handling interaction among the subparts as they are detected during the problem solving process.

Components of Planning System

The important components of planning are

1. Choosing the best rule to apply next based on the best variable available heuristic information.
2. Applying the chosen rule to compute the new problem state that arises from its application.
3. Detecting when a solution has been found.

4. Detecting dead ends so that they can be abandoned and the system's effort directed in correct direction.
5. Repairing an almost correct solution.

1. Choosing rules to apply:

- First isolate a set of differences between the desired goal state and the current state.
- Detect rules that are relevant to reduce the differences.

If several rules are found, a variety of heuristic information can be exploited to choose among them.

This technique is based on the means end analysis method.

2. Applying rules:

Applying the rules is easy.

Each rule specifies the problem state that would result from its application.

We must be able to deal with rules that specify only a small part of the complete problem state.

Different ways to do this are

Describe, for each action, each of the changes it makes the state description.

A state was described by a set of predicates representing the facts that were true in that state. Each state is represented as a predicate.

The manipulation of the state description is done using a resolution theorem prover.

3. Detecting a solution

A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transforms the initial problem state into the goal state.

Any of the corresponding reasoning mechanisms could be used to discover when a solution has been found.

4. Detecting dead ends

As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution. The same reasoning mechanisms that can be used to detect a solution can often be used for detecting a dead end.

If the search process is reasoning forward from the initial state, it can prune any path that leads to a state from which the goal state cannot be reached.

If the search process is reasoning backward from the goal state, it can also terminate a path either because it is sure that the initial state cannot be reached or little progress is being made.

In reasoning backward, each goal is decomposed into sub goals. Each of them may lead to a set of additional sub goals. Sometimes it is easy to detect that there is now way that the entire sub goals in a given set can be satisfied at once. Other paths can be pruned because they lead nowhere.

5. Repairing an almost correct solution

Solve the sub problems separately and then combine the solution to yield a correct solution. But it leads to wasted effort.

The other way is to look at the situations that result when the sequence of operations corresponding to the proposed solution is executed and to compare that situation to the desired goal. The difference between the initial state and goal state is small. Now the problem solving can be called again and asked to find a way of eliminating a new difference. The first solution can then be combined with second one to form a solution to the original problem.

When information as possible is available, complete the specification in such a way that no conflicts arise. This approach is called least commitment strategy.
It can be applied in a variety of ways.

- To defer deciding on the order in which operations can be performed.
- Choose one order in which to satisfy a set of preconditions, we could leave the order unspecified until the very end. Then we could look at the effects of each of the sub solutions to determine the dependencies that exist among them. At that point, an ordering can be chosen.

3. Explain in detail about the various machine learning methods. (June 2013)

The various types of learning are

1. Role learning
2. Learning by taking advice.
3. Learning by parameter adjustment.

4. Learning with macro-operators.
5. Chunking
6. Explanation based learning
7. Clustering
8. Correcting mistakes
9. Recording cases.
10. Managing multiple models.
11. Back propagation.
12. Reinforcement learning.
13. Genetic algorithms

1. Role learning

Rote learning is the basic learning activity. It is also called memorization because the knowledge, without any modification is, simply copied into the knowledge base. As computed values are stored, this technique can save a significant amount of time.

Rote learning technique can also be used in complex learning systems provided sophisticated techniques are employed to use the stored values faster and there is a generalization to keep the number of stored information down to a manageable level. Checkers-playing program, for example, uses this technique to learn the board positions it evaluates in its look-ahead search.

Capabilities are

1. Organized storage of information.
2. Generalization

2. Learning by taking advice

This is a simple form of learning. Suppose a programmer writes a set of instructions to instruct the computer what to do, the programmer is a teacher and the computer is a student. Once learned (i.e. programmed), the system will be in a position to do new things.

The advice may come from many sources: human experts, internet to name a few. This type of learning requires more inference than rote learning. The knowledge must be transformed into an operational form before stored in the knowledge base. Moreover the reliability of the source of knowledge should be considered.

The system should ensure that the new knowledge is conflicting with the existing knowledge. FOO (First Operational Operationaliser), for example, is a learning system which is used to learn the game of Hearts. It converts the advice which is in the form of principles, problems, and methods into effective executable (LISP) procedures (or knowledge). Now this knowledge is ready to use.

3. Learning by parameter adjustment

Here the learning system relies on evaluation procedure that combines information from several sources into a single summary static. For example, the factors such as demand and production capacity may be combined into a single score to indicate the chance for increase of production. But it is difficult to know a priori how much weight should be attached to each factor.

The correct weight can be found by taking some estimate of the correct settings and then allow the program modify its settings based on its experience. This type of learning systems is useful when little knowledge is available

In game programs, for example, the factors such as piece advantage and mobility are combined into a single score to decide whether a particular board position is desirable. This single score is nothing but a knowledge which the program gathered by means of calculation.

4. Learning with Macro-operators

Sequence of actions that can be treated as a whole are called macro-operators. Once a problem is solved, the learning component takes the computed plan and stores it as a macro-operator. The preconditions are the initial conditions of the problem just solved, and its post conditions correspond to the goal just achieved. The problem solver efficiently uses the knowledge base it gained from its previous experiences. By generalizing macro-operators the problem solver can even solve different problems. Generalization is done by replacing all the constants in the macro-operators with variables.

The STRIPS, for example, is a planning algorithm that employed macro-operators in it's learning phase. It builds a macro operator **MACROP**, which contains preconditions, post-conditions and the sequence of actions. The macro operator will be used in the future operation.

5. Chunking

Chunking is similar to learning with macro-operators. Generally, it is used by problem solver systems that make use of production systems.

A *production system* consists of a set of *rules* that are in if-then form. That is given a particular situation, what are the actions to be performed. For example, if it is raining then take umbrella.

Production system also contains knowledge base, control strategy and a rule applier. To solve a problem, a system will compare the present situation with the left hand side of the rules. If there is a match then the system will perform the actions described in the right hand side of the corresponding rule.

Problem solvers solve problems by applying the rules. Some of these rules may be more useful than others and the results are stored as a chunk. Chunking can be used to learn general search control knowledge. Several chunks may encode a single macro-operator and one chunk may participate in a number of macro sequences. Chunks learned in the beginning of problem solving, may be used in the later stage. The system keeps the chunk to use it in solving other problems.

Soar is a general cognitive architecture for developing intelligent systems. Soar requires knowledge to solve various problems. It acquires knowledge using chunking mechanism. The system learns reflexively when impasses have been resolved. An impasse arises when the system does not have sufficient knowledge. Consequently, Soar chooses a new problem space (set of states and the operators that manipulate the states) in a bid to resolve the impasse. While resolving the impasse, the individual steps of the task plan are grouped into larger steps known as chunks. The chunks decrease the problem space search and so increase the efficiency of performing the task.

In *Soar*, the knowledge is stored in long-term memory. Soar uses the chunking mechanism to create productions that are stored in long-term memory. A chunk is nothing but a large production that does the work of an entire sequence of smaller ones.

The productions have a set of conditions or patterns to be matched to working memory which consists of current goals, problem spaces, states and operators and a set of actions to perform when the production fires. Chunks are generalized before storing. When the same impasse occurs again, the chunks so collected can be used to resolve it.

6. Explanation based learning

An Explanation-based Learning (**EBL**) system accepts an example (i.e. a training example) and explains what it learns from the example. The **EBL** system

takes only the relevant aspects of the training. This explanation is translated into particular form that a problem solving program can understand. The explanation is generalized so that it can be used to solve other problems.

PRODIGY is a system that integrates problem solving, planning, and learning methods in a single architecture. It was originally conceived by Jaime Carbonell and Steven Minton, as an AI system to test and develop ideas on the role that machine learning plays in planning and problem solving. **PRODIGY** uses the **EBL** to acquire control rules.

The **EBL** module uses the results from the problem-solving trace (ie. Steps in solving problems) that were generated by the central problem solver (a search engine that searches over a problem space). It constructs explanations using an axiomatized theory that describes both the domain and the architecture of the problem solver. The results are then translated as control rules and added to the knowledge base. The control knowledge that contains control rules is used to guide the search process effectively.

7. Clustering

Discovery is a restricted form of learning. The knowledge acquisition is done without getting any assistance from a teacher. Discovery Learning is an inquiry-based learning method.

In discovery learning, the learner uses his own experience and prior knowledge to discover the truths that are to be learned. The learner constructs his own knowledge by experimenting with a domain, and inferring rules from the results of these experiments. In addition to domain information the learner need some support in choosing and interpreting the information to build his knowledge base.

A *cluster* is a collection of objects which are similar in some way. Clustering groups data items into similarity classes. The properties of these classes can then be used to understand problem characteristics or to find similar groups of data items. Clustering can be defined as the process of reducing a large set of unlabeled data to manageable piles consisting of similar items. The similarity measures depend on the assumptions and desired usage one brings to the data.

Clustering begins by doing feature extraction on data items and measure the values of the chosen feature set. Then the clustering model selects and compares two sets of data items and outputs the similarity measure between them. Clustering

algorithms that use particular similarity measures as subroutines are employed to produce clusters.

The clustering algorithms are generally classified as Exclusive Clustering, Overlapping Clustering, Hierarchical Clustering and Probabilistic Clustering. The selection of clustering algorithms depends on various criteria such as time and space complexity. The results are checked to see if they meet the standard otherwise some or all of the above steps have to be repeated.

Some of the applications of clustering are data compression, hypothesis generation and hypothesis testing. The conceptual clustering system accepts a set of object descriptions in the form of events, observations, facts and then produces a classification scheme over the observations.

COBWEB is an incremental conceptual clustering system. It incrementally adds the objects into a classification tree. The attractive feature of incremental systems is that the knowledge is updated with each new observation. In **COBWEB** system, learning is incremental and the knowledge it learned in the form of classification trees increase the inference abilities.

8. Correcting mistakes

While learning new things, there is a possibility that the learning system may make mistakes. Like human beings, learning system can correct itself by identifying reasons for its failure, isolate it, explain how the particular assumption causes failure, and modifies its knowledge base. For example, while playing chess a learning system may make a wrong move and ends up with failure. Now the learning system thinks of the reasons for the failure and corrects its knowledge base. Therefore when it plays again it will not repeat the same mistake.

In his work, Active Learning with Multiple Views, Ion Muslea has used this technique to label the data. He develops a technique known as Co-EMT which is a combination of two techniques: Co-testing and Co-EM. The Co-testing method interacts with the user to label the data. If it does any mistake in labeling, it learns from the mistakes and improves. After learning, the system labels the unlabelled data extracted from a source efficiently. The labeled data constitutes what is called knowledge.

9. Recording cases.

A program that learns by recording cases generally makes use of consistency heuristic. According to consistency heuristic, a property of something can be

guessed by finding the most similar cases from a given set of cases. For example, a computer is given the images of different types of insects, birds, and animals. If the computer is asked to identify a living thing which is not in the recorded list, it will compare the given image with already recorded ones, and will at least tell whether the given image is insect, bird or animal.Learning by recoding cases technique is mainly used in natural language learning tasks.

During the training phase, a set of cases that describe ambiguity resolution episodes for a particular problem in text analysis is collected. Each case contains a set of features or attribute-value pairs that encode the context in which the ambiguity was encountered.

Moreover, each case is annotated with solution features that explain how the ambiguity was resolved in the current example. The cases which are created are then stored in a case base. Once the training is over, the system can use the case base to resolve ambiguities in new sentences. This way, the system acquires the linguistic knowledge.

10. Managing multiple models.

Learning can be done by managing a set of general models and a set of specific models kept in a version space. A version space is nothing but a representation that is used to get relevant information from a set of learning examples.

A version space description consists of two trees which are complement to each other: one represents general model and the other represents specific model. Both positive and negative examples are used to get the two set of models converge on one just-right model. That is, positive examples generalize specific models and negative examples specialize general models. Ultimately, a correct model that matches only the observed positive examples is obtained. Query By Committee (QBC) is an algorithm which implements this technique in order to acquire knowledge.

11. Back propagation.

Back propagation is a kind of neural network. A Neural Network (or artificial neural network) is a collection of interconnected processing elements or nodes. The nodes are termed simulated neurons as they attempt to imitate the functions of biological neurons. The nodes are connected together via links. We can compare this with axon-synapse-dendrite connections in the human brain.

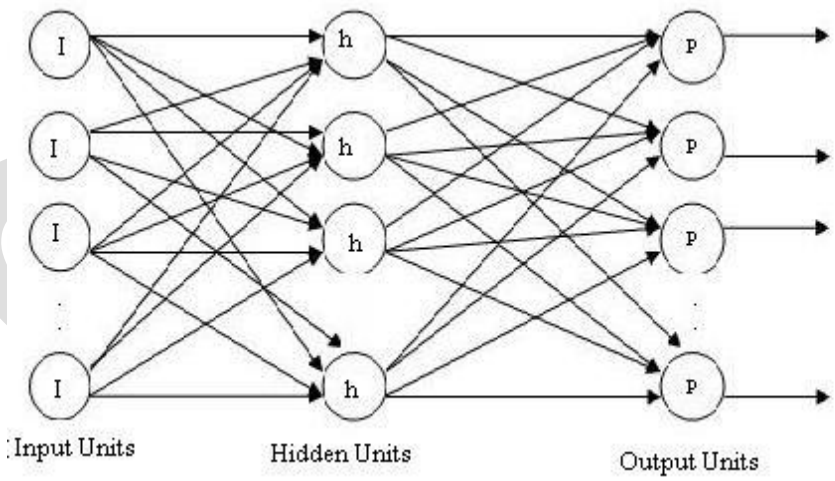
HOW BACKPROPAGATION WORKS?

Initially, a weight is assigned at random to each link in order to determine the strength of one node's influence on the other. When the sum of input values reaches a threshold value, the node will produce the output 1 or 0 otherwise. By adjusting theweights the desired output can be obtained. This training process makes the network learn. The network, in other words, acquires knowledge in much the same way human brains acquire namely learning from experience. Backpropagation is one of the powerful artificial neural network technique which is used acquire knowledge automatically.

Back propagation method is the basis for training a supervised neural network. The output is a real value which lies between 0 and 1 based on the sigmoid function. The formula for the output is,

Output = 1 / (1+e-sum).

As the sum increases, the output approaches 1. As the sum decreases, the output approaches 0.



A Multilayer Network

A multilayer network is a kind of neural network which consists of one or more layers of nodes between the input and the output nodes. The input nodes pass values to the hidden layer, which in turn passes to the output layer. A network with a layer of input units, a layer of hidden units and a layer of output units is a two-layer network. A network with two layers of hidden units is a three-layer network, and so on. The multilayer network is the basis for back propagation network.

As the name implies, there is a backward pass of error to each internal node within the network, which is then used to calculate weight gradients for that node.

The network learns by alternately propagating forward the activations and propagating backward the errors as and when they occur.

Back propagation network can deal with various types of data. It also has the ability to model a complex decision system. If the problem domain involves large amount of data, the network will require having more input units or hidden units. Consequently this will increase the complexity of the model and also increase its computational complexity. Moreover it takes more time in solving complex problems. In order to overcome this problem multi-back propagation network is used.

In the beginning of the training process, weights are assigned to the connections at random. The training process is iterative. The training cases are given to the network iteratively. The actual outputs are compared with desired outputs and the errors are calculated. The errors are then propagated back to the network in order to adjust the weights. The training process repeats till the desired outputs are obtained or the error reaches an acceptable level.

TYPES OF BACKPROPAGATION NETWORKS

Static back-propagation is one kind of back propagation networks that produces a mapping of a static input to a static output. These networks can solve static classification problems such as optical character recognition (OCR). Recurrent Back propagation is another kind of type used for fixed-point learning. NeuroSolutions, for example, is software that has this ability.

In recurrent backpropagation, activations are fed forward until a fixed value is achieved. There after the error is computed and propagated backwards. The difference between static and recurrent backpropagation is that the mapping is instantaneous in static back-propagation while it is not in the case of latter type. Moreover training a network using fixed-point learning is more difficult than with static backpropagation.

APPLICATION OF BACKPROPAGATION

The researchers, Thomas Riga, Angelo Cangelosi (University of Plymouth) and Alberto Greco (University of Genoa), have developed a model for grounding of symbol. It assumes the brain as a symbol system and explains cognition as a manipulation of symbols governed by rules.

The symbol grounding problem is nothing but connecting meaning to the symbols or images of objects received from input stimuli.

The model uses two learning algorithms: Kohonen Self-Organizing Feature Map and backpropagation algorithm.

To perform the tasks, it has two modules and a retina for input. The first module uses Kohonen Self-Organizing Feature Map and categorizes the images projected on the retina. It expresses the intrinsic order of the stimulus set in a bi-dimensional matrix known as activation matrix.

The second module relates visual input, symbolic input and output stimuli. Now it uses backpropagation algorithm for learning. The error in the output is computed with respect to the training set and is sent back to input unit and the weight distribution is corrected in order to get the correct output. In this process, the symbols which are grounded constitute the knowledge. The knowledge so acquired is then used to generate and describe new meanings for the symbols.

12. Reinforcement learning.

Reinforcement learning is one of the most active research areas in Artificial Intelligence. Reinforcement learning is training by rewards and punishments. Here we train a computer as if we train a dog. If the dog obeys and acts according to our instructions we encourage it by giving biscuits or we punish it by beating or by scolding. Similarly, if the system works well then the teacher gives positive value (i.e. reward) or the teacher gives negative value (i.e. punishment).

The learning system which gets the punishment has to improve itself. Thus it is a trial and error process. The reinforcement learning algorithms selectively retain the outputs that maximize the received reward over time. To accumulate a lot of rewards, the learning system must prefer the best experienced actions; however, it has to try new actions in order to discover better action selections for the future.

TEMPORAL DIFFERENCE LEARNING

Temporal difference learning is a central idea to reinforcement learning. It is based on Monte Carlo methods and dynamic programming.

It is an unsupervised technique. Temporal difference learning methods can learn directly from raw experience without a model of the environment's dynamics. Examples are learning to play games, robot control, elevator control, network routing and animal learning.

APPLICATIONS OF REINFORCEMENT LEARNING

Personalization Travel Support System is one software that is designed to provide travelling information as per the user's interests.

It applies the reinforcement learning to analyze and learn customer behaviors and list out the products that the customers wish to buy. If the system selects the right item that the customer wish to buy then it is given reward by assigning a particular value for the state that a user selects to perform and if the system selects an item which the user does not wish to buy then it is given the penalty.

This way the system learns the personal interests. In this process, the system acquires the knowledge of the user behavior and interest which makes it decide which information should be given to a particular user. This result in greater customer satisfaction and increase in the success rate of product promotion.

13. Genetic algorithms

Genetic algorithms are based on biological evolution. Genetic algorithms can be used to solve a wide variety of problems. Given a problem a genetic algorithm generates a set of possible solutions and evaluates each in order to decide which solutions are fit for reproduction. If a particular solution is more fit then it will have more chances to generate new solutions. Finally we can find a real solution.

Genetic algorithms are so powerful that they can exhibit more efficiency if programmed perfectly. Applications include learning Robot behavior, molecular structure optimization, automated design of mechatronic systems, and electronic circuit design.

A Genetic Approach Methodology for Knowledge Acquisition for Intelligent Diagnosis is a project done by CEEP (Center for Engineering Education and Practice), University of Michigan-Dearborn, to develop diagnosis knowledge for robot arm movements. In this project, a robot was selected for analysis. The wrong arm movements of robot were observed and collected as failure data. Then the project applied genetic algorithm to extract knowledge from the failure data. The acquisition of knowledge was automatic. The knowledge extracted is then stored in the knowledge base to make use of in the intelligent diagnosis system. The knowledge so stored is known as diagnosis knowledge as it's used to detect what went wrong and to decide the course of action in order to make the robot perfect.

4. Explain adaptive learning with an example. (November 2013)

Adaptive learning

Adaptive learning is an educational method which uses computers as interactive teaching devices, and to orchestrate the allocation of human and mediated resources according to the unique needs of each learner.

Computers adapt the presentation of educational material according to students' learning needs, as indicated by their responses to questions, tasks and experiences. The technology encompasses aspects derived from various fields of study including computer science, education, psychology, and brain science.

Adaptive learning has been partially driven by a realization that tailored learning cannot be achieved on large-scale using traditional, non-adaptive approaches. Adaptive learning systems endeavor to transform the learner from passive receptor of information to collaborator in the educational process. Adaptive learning systems' primary application is in education, but another popular application is business training. They have been designed as desktop computer applications, web applications, and are now being introduced into overall curricula.

Adaptive learning has been implemented in several kinds of educational systems such as adaptive educational hypermedia, intelligent tutoring systems, Computerized adaptive testing, and computer-based pedagogical agents, among others.

Technology and methodology

Adaptive learning systems have traditionally been divided into separate components or 'models'. While different model groups have been presented, most systems include some or all of the following models (occasionally with different names):

- Expert model - The model with the information which is to be taught
- Student model - The model which tracks and learns about the student
- Instructional model - The model which actually conveys the information
- Instructional environment - The user interface for interacting with the system

Expert model

The expert model stores information about the material which is being taught. This can be as simple as the solutions for the question set but it can also include lessons and tutorials and, in more sophisticated systems, even expert methodologies to illustrate approaches to the questions.

Adaptive learning systems which do not include an expert model will typically incorporate these functions in the instructional model.

Student model

Student model algorithms have been a rich research area over the past twenty years. The simplest means of determining a student's skill level is the method employed in CAT (Computerized adaptive testing). In CAT, the subject is presented with questions that are selected based on their level of difficulty in relation to the presumed skill level of the subject. As the test proceeds, the computer adjusts the subject's score based on their answers, continuously fine-tuning the score by selecting questions from a narrower range of difficulty.

An algorithm for a CAT-style assessment is simple to implement. A large pool of questions is amassed and rated according to difficulty, through expert analysis, experimentation, or a combination of the two. The computer then performs what is essentially a binary search, always giving the subject a question which is half way between what the computer has already determined to be the subject's maximum and minimum possible skill levels. These levels are then adjusted to the level of the difficulty of the question, reassigning the minimum if the subject answered correctly, and the maximum if the subject answered incorrectly. Obviously, a certain margin for error has to be built in to allow for scenarios where the subject's answer is not indicative of their true skill level but simply coincidental. Asking multiple questions from one level of difficulty greatly reduces the probability of a misleading answer, and allowing the range to grow beyond the assumed skill level can compensate for possible misevaluations.

Richer student model algorithms look to determine causality and provide a more extensive diagnosis of student's weaknesses by linking 'concepts' to questions and defining strengths and weaknesses in terms of concepts rather than simple 'levels' of ability. Because multiple concepts can influence a single question, questions have to be linked to all relevant concepts. For example, a matrix can list binary values (or even scores) for the intersection of every concept and every question. Then, conditional probability values have to be calculated to reflect the likelihood that a student who is weak in a particular concept will fail to correctly answer a particular question. A student takes a test, the probabilities of weakness in all concepts conditional on incorrect answers in all questions can be calculated

using Baye's Law (these adaptive learning methods are often called bayesian algorithms).

A further extension of identifying weaknesses in terms of concepts is to program the student model to analyze incorrect answers. This is especially applicable for multiple choice questions. Consider the following example:

Q. Simplify: $2x^2 + x^3$

a) Can't be simplified

b) $3x^5$

c) ...

d) ...

Clearly, a student who answers (b) is adding the exponents and failing to grasp the concept of like terms. In this case, the incorrect answer provides additional insight beyond the simple fact that it is incorrect.

Instructional model

The instructional model generally looks to incorporate the best educational tools that technology has to offer (such as multimedia presentations) with expert teacher advice for presentation methods. The level of sophistication of the instructional model depends greatly on the level of sophistication of the student model. In a CAT-style student model, the instructional model will simply rank lessons in correspondence with the ranks for the question pool. When the student's level has been satisfactorily determined, the instructional model provides the appropriate lesson. The more advanced student models which assess based on concepts need an instructional model which organizes its lessons by concept as well. The instructional model can be designed to analyze the collection of weaknesses and tailor a lesson plan accordingly.

When the incorrect answers are being evaluated by the student model, some systems look to provide feedback to the actual questions in the form of 'hints'. As the student makes mistakes, useful suggestions pop up such as "look carefully at the sign of the number". This too can fall in the domain of the instructional model, with generic concept-based hints being offered based on concept weaknesses, or the hints can be question-specific in which case the student, instructional, and expert models all overlap.

Implementation

Classroom Implementation

Adaptive learning that is implemented in the classroom environment using information technology is often referred to as an Intelligent Tutoring System or an Adaptive Learning System. Intelligent Tutoring Systems operate on three basic principles:

Systems need to be able to dynamically adapt to the skills and abilities of a student.

Environments utilize cognitive modeling to provide feedback to the student while assessing student abilities and adapting the curriculum based upon past student performance.

Inductive logic programming (ILP) is a way to bring together inductive learning and logic programming to an Adaptive Learning System. Systems using ILP are able to create hypothesis from examples demonstrated to it by the programmer or educator and then use those experiences to develop new knowledge to guide the student down paths to correct answers.

Systems must have the ability to be flexible and allow for easy addition of new content.

Cost of developing new Adaptive Learning Systems is often prohibitive to educational institutions so re-usability is essential.

School districts have specific curriculum that the system needs to utilize to be effective for the district. Algorithms and cognitive models should be broad enough to teach mathematics, science, and language.

Systems need to also adapt to the skill level of the educators.

Many educators and domain experts are not skilled in programming or simply do not have enough time to demonstrate complex examples to the system so it should adapt to the abilities of educators.

Distance learning implementation

Adaptive Learning systems can be implemented on the Internet for use in distance learning and group collaboration applications.

The field of distance learning is now incorporating aspects of adaptive learning. Initial systems without adaptive learning were able to provide automated feedback to students who are presented questions from a preselected question bank. That approach however lacks the guidance which teachers in the classroom can provide.

Current trends in distance learning call for the use of adaptive learning to implement intelligent dynamic behavior in the learning environment.

During the time a student spends learning a new concept they are tested on their abilities and databases track their progress using one of the models. The latest generation of distance learning systems take into account the students' answers and adapt themselves to the student's cognitive abilities using a concept called 'cognitive scaffolding'. Cognitive scaffolding is the ability of an automated learning system to create a cognitive path of assessment from lowest to highest based on the demonstrated cognitive abilities. A current successful implementation of adaptive learning in web-based distance learning is the Maple engine of WebLearn by RMIT university. WebLearn is advanced enough that it can provide assessment of questions posed to students even if those questions have no unique answer like those in the Mathematics field.

Group collaboration is also a hot field in the adaptive learning research area. Group collaboration is a key field in Web 2.0 which extends the functionality of distance learning. Adaptive learning can be incorporated to facilitate collaboration within distance learning environments like forums or resource sharing services. Some examples of how adaptive learning can help with collaboration include:

- Automated grouping of users with the same interests.
- Personalization of links to information sources based on the user's stated interests or the user's surfing habits.

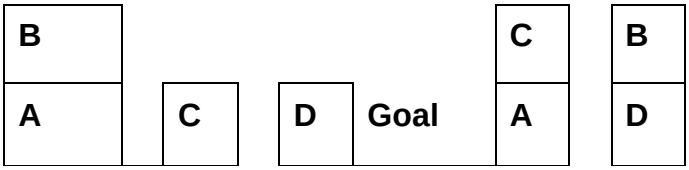
Game Design Implementation

In 2014, an educational researcher concluded a multi-year study of adaptive learning for educational game design. The research developed and validated the ALGAE (Adaptive Learning GAme dEsign) model, a comprehensive adaptive learning model based on game design theories and practices, instructional strategies, and adaptive models. The research extended previous researching in game design, instructional strategies, and adaptive learning, combining those three components into a single complex model.

The study resulted in the development of an adaptive educational game design model to serve as a guide for game designers, instructional designers, and educators with the goal of increasing learning outcomes. Survey participants validated the value of the ALGAE model and provided specific insights on the

model's construction, use, benefits, and challenges. The current ALGAE model is based on these insights. The model now serves as a guideline for the design and development of educational computer games.

5. Solve the following by goal stack planning. (June 2013)



Goal stack planning

- One of the earliest techniques is planning using goal stack.
- Problem solver uses single stack that contains
  - sub goals and operators both
  - sub goals are solved linearly and then finally the conjoined sub goal is solved.
- Plans generated by this method will contain
  - complete sequence of operations for solving one goal followed by complete sequence of operations for the next etc.
- Problem solver also relies on
  - A database that describes the current situation.
  - Set of operators with precondition, add and delete lists.

Algorithm

- Let us assume that the goal to be satisfied is:

$$GOAL = G1 \wedge G2 \wedge \ldots \wedge Gn$$

- Sub-goals G1, G2, … Gn are stacked with compound goal $G1 \wedge G2 \wedge \ldots \wedge Gn$ at the bottom.
  - Top G1
    G2
    :
    Gn
  - Bottom $G1 \wedge G2 \wedge \ldots \wedge G4$
- At each step of problem solving process, the top goal on the stack is pursued.
- Find an operator that satisfies sub goal G1 (makes it true) and replace G1 by the operator.

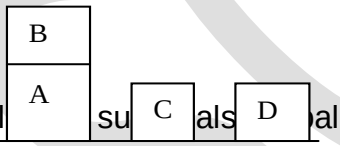
- If more than one operator satisfies the sub goal then apply some heuristic to choose one.

- In order to execute the top most operation, its preconditions are added onto the stack.
  - Once preconditions of an operator are satisfied, then we are guaranteed that operator can be applied to produce a new state.
  - New state is obtained by using ADD and DELETE lists of an operator to the existing database.
- Problem solver keeps track of operators applied.
  - This process is continued till the goal stack is empty and problem solver returns the plan of the problem.

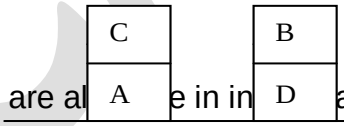
Example

- Logical representation of Initial and Goal states:
  - Initial State: ON(B, A) Λ ONT(C) Λ ONT(A) Λ ONT(D) Λ CL(B) ΛCL(C) Λ CL(D) Λ AE
  - Goal State: ON(C, A) Λ ON(B, D) Λ ONT(A) Λ ONT(D) Λ CL(C) Λ CL(B) Λ AE

Initial State



Goal State



- Following are the sub goals which are already satisfied in initial state.
  - ONT(A) Λ ONT(D) Λ CL(C) Λ CL(B) Λ AE
- Represent for the sake of simplicity - **TSUBG**.
- Only sub-goals ON(C, A) & ON(B, D) are to be satisfied and finally make sure that TSUBG remains true.
- Either start solving first ON(C, A) or ON(B, D). Let us solve first ON(C, A).

Goal Stack:

ON(C, A)

ON(B, D)

ON(C, A) Λ ON(B, D) Λ TSUBG

- To solve ON(C, A), operation S(C, A) could only be applied.
- So replace ON(C, A) with S(C, A) in goal stack.

Goal Stack:

- S (C, A)
- ON(B, D)
- ON(C, A) Λ ON(B, D) Λ TSUBG

- S(C, A) can be applied if its preconditions are true. So add its preconditions on the stack.

Goal Stack:

CL(A)

HOLD(C) Preconditions of STACK

CL(A) Λ HOLD(C)

S (C, A) Operator

ON(B, D)

ON(C, A) Λ ON(B, D) Λ TSUBG

- Next check if CL(A) is true in State_0.
- Since it is not true in State_0, only operator that could make it true is US(B, A).
- So replace CL(A) with US(B, A) and add its preconditions.

Goal Stack: ON(B, A)

CL(B) Preconditions of UNSTACK

AE

ON(B, A) Λ CL(B) Λ AE

US(B, A) Operator

HOLD(C)

CL(A)) Λ HOLD(C)

S (C, A) Operator

ON(B, D)

ON(C, A) Λ ON(B, D) Λ TSUBG

- ON(B, A), CL(B) and AE are all true in initial state, so pop these along with its compound goal.
- Next pop top operator US(B, A) and produce new state by using its ADD and DELETE lists.
- Add US(B, A) in a queue of sequence ofoperators.

SQUEUE = US (B, A)

State_1:

ONT(A) ΛONT(C) Λ ONT(D) Λ HOLD(B) ΛCL(A) Λ CL(C) ΛCL(D)

Goal Stack:

HOLD(C)

CL(A)) Λ HOLD(C)

S (C, A) Operator

ON(B, D)

ON(C, A) Λ ON(B, D) Λ TSUBG

- To satisfy the goal HOLD(C), two operators can be used e.g., PU(C) or US(C, X), where X could be any block. Let us choose PU(C) and proceed further.
- Repeat the process. Change in states is shown below.

State_1:

ONT(A) ΛONT(C) Λ ONT(D) Λ HOLD(B) ΛCL(A) Λ CL(C) ΛCL(D)

SQUEUE = US (B, A)

- Next operator to be popped of is S(B, D). So

State_2:

ONT(A) ΛONT(C) Λ ONT(D) Λ ON(B, D) ΛCL(A) Λ CL(C) ΛCL(B)ΛAE

SQUEUE = US (B, A), S(B, D)

State_3:

ONT(A) ΛHOLD(C) Λ ONT(D) Λ ON(B, D) ΛCL(A) ΛCL(B)

SQUEUE = US (B, A), S(B, D), PU(C)

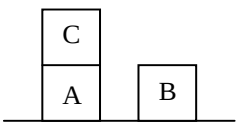
State_4:

ONT(A) ΛON(C, A) Λ ONT(D) Λ ON(B, D) ΛCL(C) ΛCL(B) ΛAE

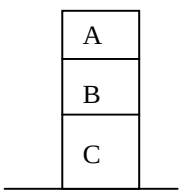
SQUEUE = US (B, A), S(B, D), PU(C), S(C, A)

- The Goal stack method is not efficient for difficult problems such as Sussman anomaly problem.
- It fails to find good solution.
- Consider the Sussman anomaly problem

Initial State (State0)



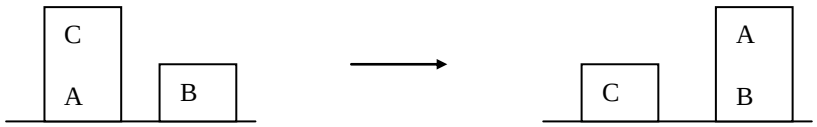
Goal State



Initial State:ON(C, A) Λ ONT(A) Λ ONT(B)

Goal State: ON(A, B) Λ ON(B, C)

- Remove CL and AE predicates for the sake of simplicity.
- To satisfy ON(A, B), following operators are applied
    - ***US(C, A) , PD(C), PU(A) and S(A, B***)

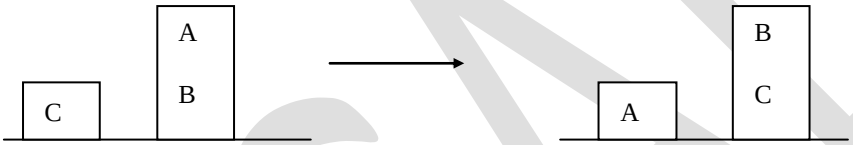


State_1: ON(B, A) Λ ONT(C)

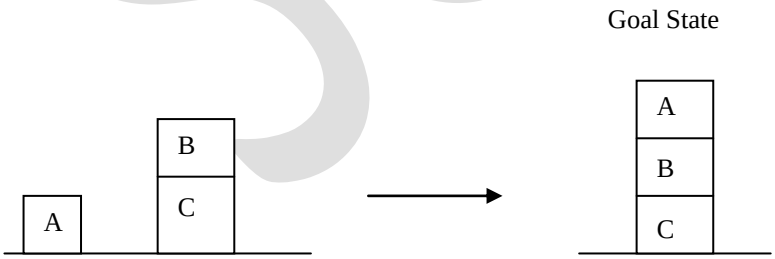
- To satisfy ON(B, C), following operators are applied

US(A, B) , PD(A), PU(B) and S(B, C)

State_2: ON(B, C) Λ ONT(A)



- Finally satisfy combined goal ON(A, B) Λ ON(B, C).
- Combined goal fails as while satisfying ON(B, C), we have undone ON(A, B).
- Difference in goal and current state is ON(A, B).
- Operations required are PU(A) and S(A, B)



- The complete plan for solution is as follows:

1. US(C, A)
2. PD (C)
3. **PU(A)**
4. **S(A, B)**
5. **US(A, B)**
6. **PD(A)**

7. PU(B)
8. S(B, C)
9. PU(A)
10. S(A, B)

- Although this plan will achieve the desired goal, but it is not efficient.
- In order to get efficient plan, either repair this plan or use some other method.
- Repairing is done by looking at places where operations are done and undone immediately, such as S(A, B) and US(A, B).
- By removing them, we get
  1. US(C, A)
  2. PD (C)
  3. PU(B)
  4. S(B, C)
  5. PU(A)
  6. S(A, B)

6. Solve the blocks world problem using strips. How does it act as a planning system?

(May 2014) (MAY/JUNE 2016)

Block World Problem

- Block world problemassumptions
  - Square blocks of same size
  - Blocks can be stacked one upon another.
  - Flat surface (table) on which blocks can be placed.
  - Robot arm that can manipulate the blocks. It can hold only one block at a time.
- In block world problem, the state is described by a set of predicates representing the facts that were true in that state.
- One must describe for every action, each of the changes it makes to the state description.
- In addition, some statements that everything else remains unchanged is also necessary.

Actions (Operations) done by Robot

- UNSTACK (X, Y) : **[US (X, Y)]**
  - Pick up X from its current position on block Y. The arm must be empty and X has no block on top of it.
- STACK (X, Y): **[S (X, Y)]**
  - Place block X on block Y. Arm must holding X and the top of Y is clear.
- PICKUP (X): **[PU (X)]**
  - Pick up X from the table and hold it. Initially the arm must be empty and top of X is clear.
- PUTDOWN (X): **[PD (X)]**
  - Put block X down on the table. The arm must have been holding block X.
- Predicates used to describe the state
  - ON(X, Y) - Block X on block Y.
  - ONT(X) - Block X on the table.
  - CL(X) - Top of X clear.
  - HOLD(X) - Robot-Arm holding X.
  - AE - Robot-arm empty.
- Logical statements true in this block world.
  - Holding X means, arm is not empty
    - $(\exists X)$ HOLD (X) $\rightarrow \sim$ AE
  - X is on a table means that X is not on the top of any block
    - $(\forall X)$ ONT (X) $\rightarrow \sim (\exists Y)$ ON (X, Y)
  - Any block with no block on has clear top
    - $(\forall X)(\sim (\exists Y)$ ON (Y,X)) $\rightarrow$ CL (X)
- ***Effect of Unstack operation***
- The effect of US(X, Y) is described by the following axiom
    - [CL(X, State) Λ ON(X, Y, State)] $\rightarrow$
      - [HOLD(X, DO(US (X, Y), State)) Λ CL(Y, DO(US(X, Y), State))]
  - DO is a function that generates a new state as a result of given action and a state.

- For each operator, set of rules (called frame axioms) are defined where the components of the state are
    - affected by an operator
        - If US(A, B) is executed in state S0, then we can infer that HOLD (A, S1) Λ CLEAR (B, S1) holds true, where S1 is new state after Unstack operation is executed.
    - not affected by an operator
- If US(A, B) is executed in state S0, B in S1 is still on the table but we can't derive it. So frame rule stating this fact is defined as ONT(Z, S) $\rightarrow$ ONT(Z, DO(US (A, B), S))
- Advantage of this approach is that
    - simple mechanism of resolution can perform all the operations that are required on the state descriptions.
- Disadvantage is that
    - number of axioms becomes very large for complex problem such as COLOR of block also does not change.
    - So we have to specify rule for each attribute.
        - COLOR(X, red, S) $\rightarrow$
            - COLOR(X, red, DO(US(Y, Z), s))
- To handle complex problem domain, there is a need of mechanism that does not require large number of explicit frame axioms.

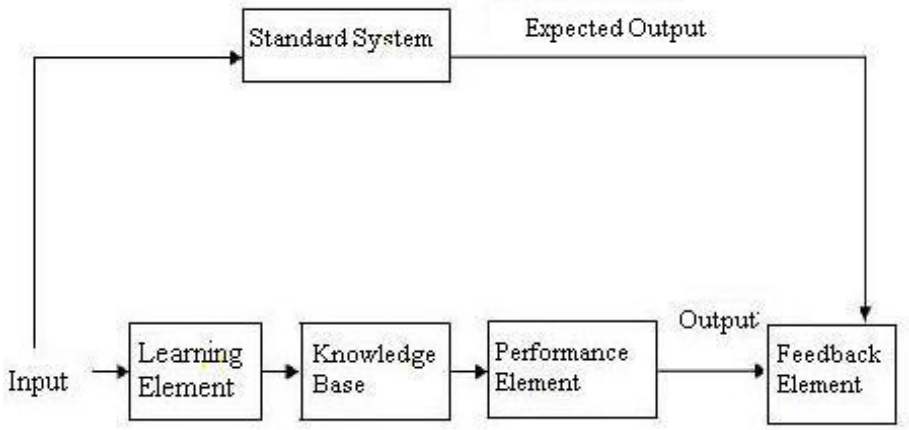
7. i) Explain the steps in designing a learning system. (May 2014)

ii) What is ID3? Write the drawback of ID3. (May 2016)

To solve problems computers require intelligence. Learning is central to intelligence. As intelligence requires knowledge, it is necessary for the computers to acquire knowledge. Machine learning serves this purpose.

Machine learning refers to a system capable of acquiring and integrating the knowledge automatically. The capability of the systems to learn from experience, training, analytical observation, and other means, results in a system that can continuously self-improve and thereby exhibit efficiency and effectiveness.

A machine learning system usually starts with some knowledge and a corresponding knowledge organization so that it can interpret, analyze, and test the knowledge acquired.



The figure shown above is a typical learning system model. It consists of the following components.

1. Learning element.
2. Knowledge base.
3. Performance element.
4. Feedback element.
5. Standard system.

1. Learning element

It receives and processes the input obtained from a person (i.e. a teacher), from reference material like magazines, journals, etc, or from the environment at large.

2. Knowledge base

This is somewhat similar to the database. Initially it may contain some basic knowledge. Thereafter it receives more knowledge which may be new and so be added as it is or it may replace the existing knowledge.

3. Performance element

It uses the updated knowledge base to perform some tasks or solves some problems and produces the corresponding output.

4. Feedback element

It is receiving the two inputs, one from learning element and one from standard (or idealized) system. This is to identify the differences between the two

inputs. The feedback is used to determine what should be done in order to produce the correct output.

5. Standard system

It is a trained person or a computer program that is able to produce the correct output. In order to check whether the machine learning system has learned well, the same input is given to the standard system. The outputs of standard system and that of performance element are given as inputs to the feedback element for the comparison. Standard system is also called idealized system.

The sequence of operations described above may be repeated until the system gets the desired perfection.

There are several factors affecting the performance.

They are,

- Types of training provided.
- The form and extent of any initial background knowledge.
- The type of feedback provided.
- The learning algorithms used.

Training is the process of making the system able to learn. It may consist of randomly selected examples that include a variety of facts and details including irrelevant data. The learning techniques can be characterized as a search through a space of possible hypotheses or solutions. Background knowledge can be used to make learning more efficient by reducing the search space. The feedback may be a simple yes or no type of evaluation or it may contain useful information describing why a particular action was good or bad. If the feedback is always reliable and carries useful information, the learning process will be faster and the resultant knowledge will be correct.

The success of machine learning system also depends on the algorithms. These algorithms control the search to find and build the knowledge structures. The algorithms should extract useful information from training examples.

ii) What is ID3? Write the drawback of ID3. (May 2016)

- ID3 (Iterative Dichotomiser 3)
- In decision tree learning, ID3 is an algorithm used to generate a decision tree from a dataset.
- ID3 is the precursor to the C4.5 algorithm, and is typically used in the machine learning and natural language processing domains.

Algorithm:

1. Calculate the entropy of every attribute using the data set
2. Split the set S into subsets using the attribute for which entropy is minimum
3. Make a decision tree node containing that attribute
4. Recurse on subsets using remaining attributes.

Disadvantages of using ID3

- Data may be over-fitted or over-classified, if a small sample is tested.
- Only one attribute at a time is tested for making a decision.
- Classifying continuous data may be computationally expensive, as many trees must be generated to see where to break the continuum.

UNIT V EXPERT SYSTEMS

Expert systems - Architecture of expert systems, Roles of expert systems - Knowledge Acquisition –Meta knowledge, Heuristics. Typical expert systems - MYCIN, DART, XOON, Expert systems shells.

PART A

1. What are expert systems?

An expert system is an interactive computer-based decision tool that uses both facts and heuristics to solve difficult decision making problems, based on knowledge acquired from an expert.

An expert system is a model and associated procedure that exhibits, within a specific domain, a degree of expertise in problem solving that is comparable to that of a human expert.

An expert system compared with traditional computer:

Inference engine + Knowledge = Expert system

(Algorithm + Data structures = Program in traditional computer)

First expert system, called DENDRAL, was developed in the early 70's at Stanford University.

2. What is the difference between conventional system and expert system?

Conventional System	Expert System
Knowledge and processing are combined in one programme.	Knowledge database and the processing mechanism (inference) are two different components.
Programme does not make errors (only Programming error).	The ES programme may be make a mistake.
Usually it will not explain why the data needs to be input or how the decision is achieved.	Explanation is part of an ES component
System is operational only when fully developed.	An ES can operate with small amount of rules.
Step by step execution according to fixed algorithms is necessary.	Execution done logically and heuristically.
Needs complete and full information.	Can operate with sufficient or insufficient information.
Manipulates a large and effective database.	Manipulates a big and effective database.
Referencing and use of data.	Referencing and use of Knowledge.
Main objective is efficiency.	Main objective is effectiveness.
Easily operated with quantitative data.	Easily operated with qualitative data

3. Define heuristics.

Heuristic is a rule of thumb that probably leads to a solution. Heuristics play a major role in search strategies because of exponential nature of the most problems. Heuristics help to reduce the number of alternatives from an exponential number to a polynomial number.

4. Define Meta Knowledge and its use. (December 2011, November 2013) (MAY/JUNE 2016)

- Meta knowledge includes the ability to evaluate the knowledge available, the additional knowledge required and the systematic implied by the present rules.

Jacques Pitrat definition:

- Meta-knowledge is knowledge about knowledge, rather than knowledge from a specific domain such as mathematics, medicine or geology.
- According to this definition, meta-knowledge is at the heart of the learning process, which consists in transforming information into knowledge:
    - By attributing values to knowledge from other domains: truth, usefulness, importance, knowledge priority, competence of an individual towards a knowledge object, etc.
    - By describing « intellectual acts », processes that facilitate knowledge processing in other domains: memorization, understanding, application, analysis, synthesis, evaluation, etc.
    - by representing strategies to acquire, process and use knowledge from other domains : memorization techniques, heuristic principles for problem solving, project management strategies, etc.

5. List the major uses of expert system.

The Expert System is used to:

- Help experts in their routine to improve productivity.
- Help experts in some of their more complex and difficult tasks so that the problem can be managed effectively.
- Help an expert to obtain information needed by other experts who have forgotten about it or who are too busy to search for it.

6. What is the use of expert system tools?

Expert system tool is extremely expensive and powerful. The advantage of using this tool is that it provides a variety in knowledge representation techniques such as rules and frames.

7. Who is fit to be called an expert?

Anyone can be called an expert as long as that person has a vast knowledge of the particular field and has practical experience in a certain domain. However, the person is restricted to his or her own domain.

For example, being an IT expert does not mean that the person is an expert in all IT domains but she may be an expert in intelligence systems or an expert in only the development of an intelligence agent.

8. Name the programming methods supported by expert system tools?

An ES can be developed using a symbolic language such as LISP or PROLOG, or a conventional higher-level language such as FORTRAN, C and PASCAL.

LISP

All ES developed in the early days used LISP, or tools written using the LISP language.

PROLOG

The on-going research of artificial intelligent has given birth to the programming language PROLOG. PROLOG is the acronym for 'Programming in Logic'. A programme using PROLOG can be assumed to be a knowledge database that stores facts and rules.

9. Name some expect systems developed for voice recognition and their uses.

Hearsay was an early attempt at solving voice recognition through an expert systems approach. Hearsay and all interpretation systems are essentially pattern recognition systems—looking for patterns in noisy data. In the case of Hearsay recognizing phonemes in an audio stream. Other early examples were analyzing sonar data to detect Russian submarines. These kinds of systems proved much more amenable to a neural network AI solution than a rule-based approach.

10. Name some Expert systems used in Medicine.

CADUCEUS and MYCIN were medical diagnosis systems. The user describes their symptoms to the computer as they would to a doctor and the computer returns a medical diagnosis.

11. How is Expert system used to study organic molecules?

Dendral was a tool to study hypothesis formation in the identification of organic molecules. The general problem it solved—designing a solution given a set of constraints—was one of the most successful areas for early expert systems applied to business domains such as sales people configuring Dec Vax computers and mortgage loan application development.SMH.PAL is an expert system for the assessment of students with multiple disabilities.

12. How is expert system used to monitor dam safety and landslides?

Mistral is an expert system for the monitoring of dam safety developed in the 90's by Ismes (Italy). It gets data from an automatic monitoring system and performs a diagnosis of the state of the dam. Its first copy, installed in 1992 on the Ridracoli Dam (Italy), is still operational 24/7/365. It has been installed on several dams in Italy and abroad (e.g. Itaipu Damin Brazil), as well as on landslides under the name of Eydenet, and on monuments under the name of Kaleidos. Mistral is a registered trade mark of CESI.

13. Distinguish between Ignorance and uncertainty.

Ignorance does not know something that is knowable. Uncertainty is where something is not knowable: it is inherent in the situation. Ignorance can come from (a) the limited knowledge of human expert; (b) inexact data; or (c) incomplete data (which forces a premature decision).

14. When is an Expert System Appropriate?

An expert system is appropriate.

- Need justifies cost and effort
- Human expertise not always available
- Problem requires symbolic reasoning
- Problem domain is well structured
- Traditional computing methods fail
- Cooperative and articulate experts exist
- Problem is not too large

15. Write any four earliest expert systems. (MAY/JUNE 2016)

- MYCIN
- DART
- XOON
- Expert System Shell

PART B

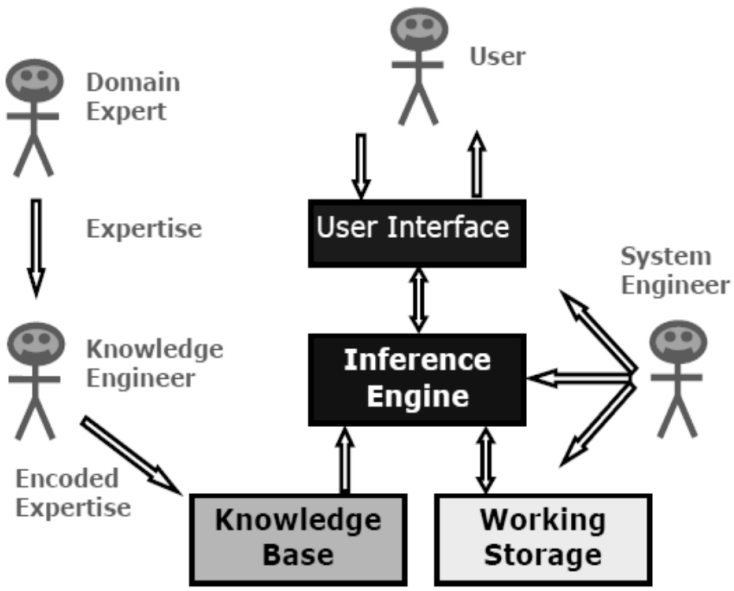
1. Draw the schematic of expert system and explain. (December 2011, November 2012, June 2013) / Explain the basic components of expert systems. (MAY/JUNE 2016)

Expert systems have a number of major system components and interface with individuals who interact with the system in various roles.

I. Components and Interfaces

1. Knowledge base :

Expert system is built around a knowledge base module. Expert system contains a formal representation of the information provided by the domain expert. This information may be in the form of problem-solving rules, procedures, or data intrinsic to the domain. To incorporate these information into the system, it is necessary to make use of one or more knowledge representation methods. Transferring knowledge from the human expert to a computer is often the most difficult part of building an expert system.



Components of Expert System

The knowledge acquired from the human expert must be encoded in such a way that it remains a faithful representation of what the expert knows, and it can be manipulated by a computer.

Three common methods of knowledge representation evolved over the years are

1. IF-THEN rules.
2. Semantic networks.
3. Frames.

1. IF-THEN rules

Human experts usually tend to think along :

condition action or Situation conclusion

Rules **"if-then"** are predominant form of encoding knowledge in expert systems. These are of the form :

If a1 , a2 , ... , an Then b1 , b2 , ... , bn where

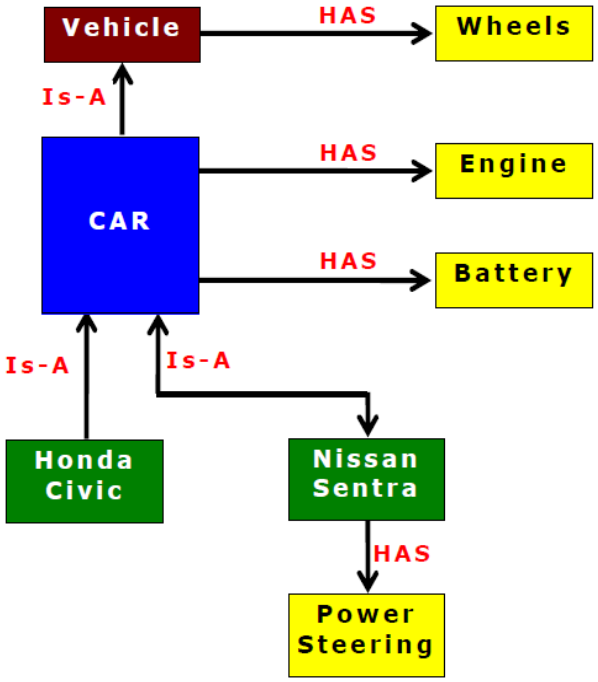
each **ai** is a condition or situation, and each **bi** is an action or a conclusion.

2.Semantic Networks

In this scheme, knowledge is represented in terms of objects and relationships between objects. The objects are denoted as nodes of a graph. The relationship between two objects are denoted as a link between the corresponding two nodes. The most common form of semantic networks uses the links between nodes to represent **IS-A** and **HAS** relationships between objects.

Example of Semantic Network

The Fig. below shows a car **IS-A** vehicle; a vehicle **HAS** wheels. This kind of relationship establishes an inheritance hierarchy in the network, with the objects lower down in the network inheriting properties from the objects higher up.



3. Frames

In this technique, knowledge is decomposed into highly modular pieces called frames, which are generalized record structures. Knowledge consists of concepts, situations, attributes of concepts, relationships between concepts, and procedures to handle relationships as well as attribute values.

Each concept may be represented as a separate **frame**.

The attributes, the relationships between concepts, and the procedures are allotted to **slots** in a frame.

The contents of a slot may be of any **data type** - numbers, strings, functions or procedures and so on.

The frames may be linked to other frames, providing the same kind of inheritance as that provided by a semantic network.

A frame-based representation is ideally suited for objected-oriented programming techniques.

Example : Frame-based Representation of Knowledge.

Two frames, their slots and the slots filled with data type are shown.

Frame	Car
Inheritance Slot	Is-A
Value	Vehicle
Attribute Slot	Engine
Value	Vehicle
Value	1
Value	
Attribute Slot	Cylinders
Value	4
Value	6
Value	8
Attribute Slot	Doors
Value	2
Value	5
Value	4

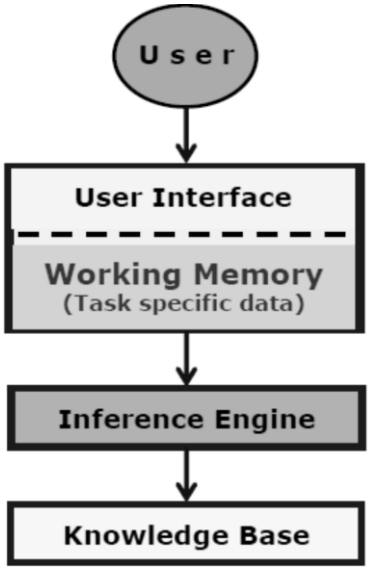
Frame	Car
Inheritance Slot	Is-A
Value	Car
Attribute Slot	Make
Value	Honda
Value	
Value	
Attribute Slot	Year
Value	1989
Value	
Value	
Attribute Slot	
Value	
Value	
Value	

2. Working storage

Working memory refers to task-specific data for a problem. The contents of the working memory, changes with each problem situation. Consequently, it is the most dynamic component of an expert system, assuming that it is kept current.

- Every problem in a domain has some unique data associated with it.
- Data may consist of the set of conditions leading to the problem, its parameters and so on.
- Data specific to the problem needs to be input by the user at the time of using, means consulting the expert system. The Working memory is related to user interface

Fig. below shows how Working memory is closely related to user interface of the expert system.



3. Inference engine :

The code at the core of the system which derives recommendations from the knowledge base and problem specific data in working storage;

The inference engine is a generic control mechanism for navigating through and manipulating knowledge and deduce results in an organized manner. The inference engine's generic control mechanism applies the axiomatic (self-evident) knowledge present in the knowledge base to the task-specific data to arrive at some conclusion.

- Inference engine the other key component of all expert systems.
- Just a knowledge base alone is not of much use if there are no facilities for navigating through and manipulating the knowledge to deduce
- something from knowledge base.A knowledge base is usually very large, it is necessary to have inferencing mechanisms that search through the database and deduce results in an organized manner.

The Forward chaining, Backward chaining and Tree searches are some of the techniques used for drawing inferences from the knowledge base.

4. User interface :

The code that controls the dialog between the user and the system.

II. Roles of Individuals who interact with the system

1. Domain expert : The individuals who currently are experts in solving the problems; here the system is intended to solve;

2. Knowledge engineer : The individual who encodes the expert's knowledge in a declarative form that can be used by the expert system;

3. User : The individual who will be consulting with the system to get advice which would have been provided by the expert.

III. Expert System Shells

Many expert systems are built with products called expert system shells. A shell is a piece of software which contains the user interface, a format for declarative knowledge in the knowledge base, and an inference engine. The knowledge and system engineers use these shells in making expert systems.

1. Knowledge engineer: uses the shell to build a system for a particular problem domain.

2. System engineer: builds the user interface, designs the declarative format of the knowledge base, and implements the inference engine.

Depending on the size of the system, the knowledge engineer and the system engineer might be the same person.

2. Write short notes on DART and MYCIN (May 2012, November 2012, June 2013, November 2013, May 2014)

i)DART

It is an artificial intelligence based expert system used for computer system fault diagnosis. This system is an automated consultant that advises IBM field service personnel on the diagnosis of faults occurring in computer installations.

The consultant identifies specific system components (both hardware and software) likely to be responsible for an observed fault and offers a brief explanation of the major factors and evidence supporting these indictments.

The consultant, called DART, was constructed using HMYCIN, and is part of a larger research effort investigating automated diagnosis of machine faults.

Motivation and Scope of Effort

A typical, large-scale computer installation is composed of numerous subsystems including CPUs, primary and secondary storage, peripherals, and supervisory software. Each of these subsystems, in turn, consists of a richly connected set of both hardware and software components such as disk drives, controllers, CPUs, memory modules, and access methods. Generally, each individual component has an associated set of diagnostic aids designed to test its own specific integrity.

However, very few maintenance tools and established diagnostic strategies aimed at identifying faults on the system or subsystem level.

As a result, identification of single or multiple faults from systemic manifestations remains a difficult task. The non-specialist field service engineer is trained to use the existing component-specific tools and, as a result, is often unable to attack the failure at the systemic level. Expert assistance is then required, increasing both the time and cost required to determine and repair the fault.

The design of DART reflects the expert's ability to take a systemic viewpoint on problems and to use that viewpoint to indict specific components, thus making more effective use of the existing maintenance capabilities.

For our initial design, the concentration was on problems occurring within the teleprocessing (TP) subsystems for the IBM 370-class computers. This subsystem includes various network controllers, terminals, remote-job entry facilities, modems, and several software access methods. In addition to these well-defined components there are numerous available test points the program can use during diagnosis. The focus was on handling two of the most frequent TP problems, (1) when a user is unable to log on to the system from a remote terminal, and (2) when the system operator is unable to initialize the TP network itself. In a new system configuration, these two problems constitute a significant percentage of service calls received.

Interviews with field-service experts made it apparent that much of their problem-solving expertise is derived from their knowledge of several well-defined communications *protocols.* Often composed of simple request-acknowledge sequences, these protocols represent the transactions between components that are required to perform various TP tasks. Although based on information found in reference manuals it is significant that these protocols are not explicitly detailed anywhere in the standard maintenance documentation. Knowledge of the basic contents of these protocols and their common sequence forms the basis of a diagnostic strategy: use the available tracing facilities to capture the actual communications occurring in the network, and analyze this data to determine which link in the protocol chain has broken. This procedure is sufficient to identify specific faulty components in the network.

The DART Consultation

During a DART consultation session, the field engineer focuses on a particular computer system that is experiencing a problem. Many installations arc

composed of numerous CPUs that partially share peripherals, thus, the term "system" is taken to mean a single CPU-complex and its attached peripherals. Within each such system, the user describes one or more problems by indicating a failure symptom, currently using a list of keywords. Using this description, the consultant makes an initial guess about which of the major subsystems might be involved in the fault. The user is then given the opportunity to select which of these implicated subsystems are to be pursued and in which order.

Each subsystem serves as a focal point for tests and findings associated with that segment of the diagnostic activity. These subsystems currently correspond to various input/output facilities (e.g.. DISK. TAPE. TP) or the CPU-complex itself, lor each selected subsystem, the user is asked to identify one or more *logical pathways* which might be involved in the situation. Kach of these logical pathways correspond to a line of communication between a peripheral and an application program. On the basis of this information and details of the basic composition of the network, the appropriate communications protocol can be selected. The user is also asked to indicate which diagnostic tools (e.g., traces, dumps, logic probes) arc available for examining each logical pathway.

Once the logical pathway and protocol have been determined, descriptions arc gathered of the often multiple *physical pathways* that actually implement the logical pathway, it is on this level that diagnostic test results arc presented and actual component indictments occur. For DART to be useful at this level, the field engineer must be familiar with the diagnostic equipment and software testing and tracing facilities which can be requested, and, of course, must also have access to information about the specific system hardware and softwate configuration of the installation. Finally, at the end of the consultation session, DART summarizes its findings and recommends additional tests and procedures to follow. Figure below depicts the major steps of the diagnostic process outlined above.

Report Findings and Recommendations for the Session
Determine Protocol Violations and Make Specific Component Indictments for each Physical Pathway
Select Appropriate Protocol and Available Diagnostic Tools for each logical Pathway
Identify Logical Pathways for each Subsystem
Infer Suspected CPU-I/O Subsystems for each Problem
Gather Initial Symptom and Configuration Information for each System

The DART Diagnostic Inference Process

After DART has indicated the components most likely to be at fault, the responsibility for performing a detailed determination and repair of the actual component failures (i.e.. microcode bugs, integrated circuit failure, etc.) would then shift to the appropriate maintenance groups for those components.

Result

The current DART knowledge base consists of 300 EMYCIN parameters and 190 production rules and was constructed over a period of 8 months. During this period 5 specialists were interviewed about different aspects of the diagnostic process and the knowledge base reflects their composite expertise. Much of the requested diagnostic data is already in a machine-readable form on the subject computer system. However, as the transcript shows, this information must currently be entered by the user. This interaction forms a substantial fraction of me users input. Indeed, wc estimate 30 to 60 percent of the current interaction between program and user will be eliminated when this on-line data is exploited.

It is clear that the communications protocols form the crux of the expertise, both for the human specialist and for DART. Although the experts were able to easily

articulate these protocols, their translation into the production rule formalism was tedious. A protocol represents an expected sequence of transactions between components. However, in order to identify specific faulty components, the production rules capture only the possible *deviations* from this expected sequence. Thus each protocol yields a substantial number of rules which only indirectly reflect the original sequence and tend to produce rather opaque explanations of the diagnostic reasoning. Furthermore, for any lengthy protocol, ensuring the *completeness* of the resulting rule set becomes a significant problem. In a collateral effort, wc are investigating the use of explicit representations of these protocols with general diagnostic rules which will hypothesize deviations directly from the protocols.

ii. MYCIN.

MYCIN was developed at Stanford university by Shortliffe (1970's). It was an expert system for diagnosing blood diseases. It was a precursor to today's expert systems and acts as an ideal case study.

Features

MYCIN is a rule based ES using backward chaining.

A typical rule

IF stain of organism is gram negative
AND morphology is rod
AND is anaerobic
THEN suggestive that class is enterobacteriaceae

The rules were actualyl stored as LISP expressions.

Inexact reasoning was employed using certainty factors. This is a number on the scale -1 to 1. -1 being definitely false +1 definitely true. MYCIN used meta-rules to redirect search at stages.

An example of such a meta rule is

IF infection is pelvic abscess
AND rules mention in premise E
AND rules mention in premise gram pos rods
THEN evidence should use rules for E before rules for gram pos rods

The above is guiding the search by saying which rules to try first.

To make the system acceptable to users (doctors) MYCIN incorporated Natural Language i.e Users entered information in English and were answered in English. A

spelling checker made intelligent corrections to the input. Both WHY, WHY NOT and HOW explanation facilities were provided. Several suggestions were often offered listed with some priority, allowing the doctor some freedom of selection.

The Decision Process

There are four key questions in the process of deciding on treatment:

1. Does the patient have a significant infection?
2. What are the organism(s) involved?
3. What drugs might be work to treat the infection?
4. What is the best choice of drug or combination of drugs to treat the infection?

MYCIN Components

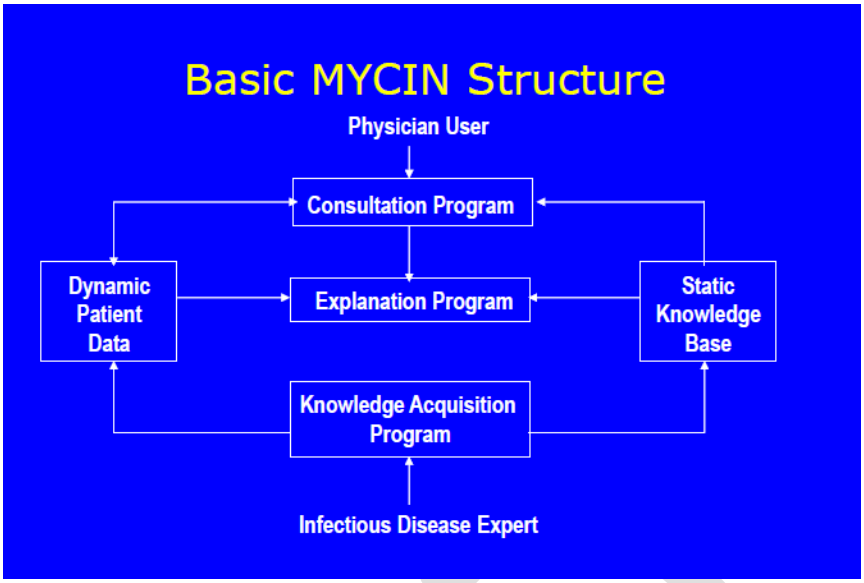
There are 5 basic components present in mycin. They are

1. KNOWLEDGE BASE: facts and knowledge about the domain
2. DYNAMIC PATIENT DATABASE: information about a particular case
3. CONSULTATION MODULE: asks questions, gives advice on a particular case
4. EXPLANATION MODULE: answers questions and justifies advice
5. KNOWLEDGE ACQUISITION PROGRAM: adds new rules and changes existing rules

Phases of MYCIN

MYCIN has two phases in its approach, a diagnosis and a prescription phase. In the first phase the nature of the infection and the organism causing the infection are determined (hopefully). The prescription phase then indicates the drugs for the treatment taking into account any side effects they may have on the patient.

MYCIN consists of about 500 rules. It is a backward chaining rule based system using a depth-first search strategy.



Evaluation

In order to assess the performance of MYCIN a set of ten case histories of meningitis were selected. The results of MYCIN were compared with medics of various degrees of expertise from Stanford. They were also compared with evaluations from experts outside of Stanford (the evaluators). They were to classify the prescriptions according to the categories:

1. Equivalent identical or equivalent recommendation.
2. Acceptable alternative different but also considered correct.
3. Unacceptable considered incorrect or inappropriate.

The evaluations were carried out in a double blind situation.

The results are

Prescriber	%Acceptable	No. Wrong
MYCIN	70	0
Prior Rx	70	0
Faculty 1	50	1
Faculty 2	50	1
Fellow	50	1
Faculty 4	50	0
Faculty 3	40	0
Resident	30	1
Faculty5	30	0
Student	10	3

3. Explain the pitfalls in using an expert system. (May 2012)

Several weaknesses concerning the use of ES.

(a) Not Widely Used

ES is not widely used in business firms or organisations. Due to limited usage, firms are still in doubt about the capability and, most definitely, the high cost involved in investing in an ES.

(b) Difficult to Use

Using an ES is very difficult and learning and mastering it requires a long time. This discourages managers from using ES. In one aspect, developing an ES that is user-friendly is the biggest challenge for ES developer.

(c) Limited Scope

This is the most obvious weakness in an ES; its scope is very limited to its field only. In the development aspect, the ES built is best developed because of its high accuracy. However, usage-wise decision makers face constantly changing problems which involve different fields that are interrelated.

(d) Probable Decision Error

The main source of the knowledge is experts. Humans make mistakes. If the experts input wrong information into the Expert system, this will have a negative impact on the results produced.

(e) Difficult to Maintain

The information in ES must be constantly updated to solve new problems. Every new problem that occurs needs new knowledge and expertise. This means that there must be an on-going relationship between the domain experts and the ES developer. This situation requires the domain experts update the source of knowledge and expertise to suit the current situation.

(f) Costly Development

The cost to consult a group of experts is not cheap, what if ES was built traditionally without involving the use of an Expert System shell? On the other hand, programming cost is high because the artificial intelligence technique is difficult to master and needs a very skilful programmer.

(g) Legal and Ethical Dilemma

We must be responsible for our actions and decisions. An expert has to take responsibility for the information he or she provides. . The difficult question here

is who should shoulder the responsibility if a decision suggested by ES results in a negative outcome.

4. Explain in detail about XCON. (November 2012)

This system is an Expert configure.

Stages of Expert System building

There are 6 stages in expert system building. They are

1. Identification
2. Conceptualization
3. Formalization
4. Implementation
5. Testing and evaluation.
6. Maintenance

1. Identification: this phase is used to identify the problems, data, goals, company, people…

- DEC, Digital Equipment Corporation Large computer manufacturer, started in 1957
- Catalogue has 40.000 different parts
- Buyer (with Sales Rep) sends order, typically 100 parts
- Delivery and assembly by DEC personnel
- Too often, part collection does not allow installation
- Too often, installed computer does not meet requirements
- Remedy: Completely assemble and test system in factory
- Automate configuration problem attempts with procedural languages were unsuccessful
- XS approach started around 1980

2. Conceptualization: this phase is used to characterize different kinds of concepts and relations
3. Formalization: it is used to express character of search
   - Configuration engineers could talk well to Knowledge Engineers of the CSDG
   - Could explain in what stage which component should be configured how

- This was expressed in production rules
  IF c1, c2 c3 THEN a1, a2, a3
- Configuration stage was explicitly represented as data: current goal or context
- Changing contexts moved configuration process through all stages

4. Implementation:
   Build the system in executable form
   - Language: OPS5 (similar to CLIPS)
   - Conflict Resolution: MEA (extends Lex / Specificity)
   - Means-Ends Analysis: order by recency of first condition
     IF c1, c2 THEN .. is now different from IF c2, c1 THEN
   - Contexts are treated as special by putting them first
   - End-task is unspecific, thus executed last
   - Use MEA + Spec to concentrate on subtasks:
     - IF g1, x, y THEN assert barify // Signal necessity of subtask
     - IF barify, a THEN p, q // Two rules perform the task
     - IF barify, b THEN r, s // of barification per se
     - IF barify THEN retract barify // Termination when ready
5. Testing and Evaluation : Does it do what we wanted?
   - Field test after 1 year, production after another year
   - Accuracy over 95%:*No more pre-assembly was necessary!*
   - The installed configurations are optimized:*Buyers are happier because they see better products*Less retraining of staff on product changes:*Quicker change of production*
   - Net return to Digital is estimated to 40M$ per year.
6. Maintenance
   Adapt to changing environment or requirements

Biological components

1. Users of XCON

- Sales: Use for quotations and ensure technical validity of orders (XSEL)
- Manufacturing: Check installability of order, guide assembly instructions and diagnostics

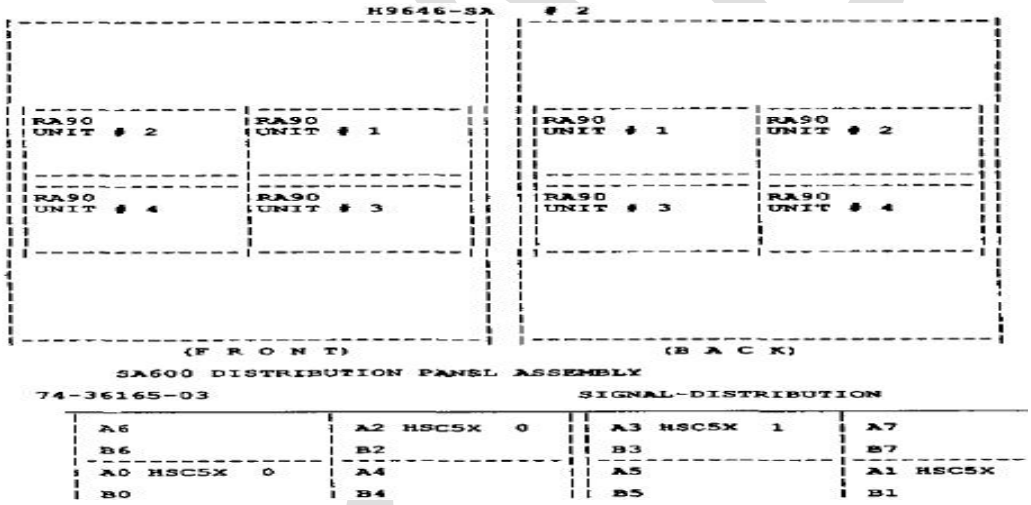
- Field service: Easy assembly at customer's site (XFL)
- Development: Anticipate integration problems for new products

The system found more use than what it was designed for

2. Key Roles

- Champion: Vision, belief in technology, influence on sponsor.
- Sponsor: Has interest in problem and control over resources: money & people
- Manager: Keeps parties together, goals realistic
- Knowledge Engineer: AI and Knowledge
- Software Engineer: Manage traditional parts, versions, architecture
- Experts: Provide domain knowledge
- Users: Feedback on fit in business process

Data set of XCON



Five data bases:

1. Components: Holds 30k parts, 25 to 125 attributes
2. Container: Extra data on backplanes and cabinets, size and slots.
3. Diagrams: 1200 ASCII arts for use in output diagrams
4. Questions: Motivate and ask data from user, eg. *Do you already have …?*
5. Configuration: Consultation data like input and diagrams

Impact of RIME/XCON on later developments

- Use of Elicitation Tools has become common practice
- Such tools can bridge the gap between knowledge and their implementation as rule (cp. Façade templates)

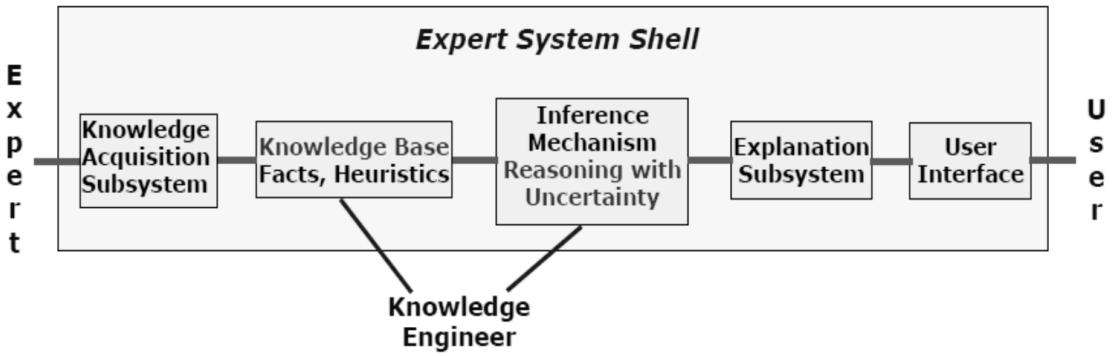
- Knowledge *structure model* can be made explicit in the tool and guide the knowledge elicitation by providing the expert with questions in a context
- Success stimulated XS applications

5. Explain in detail about expert system shells. (June 2013)

An Expert system shell is a software development environment. It contains the basic components of expert systems. A shell is associated with a prescribed method for building applications by configuring and instantiating these components.

Shell components and description

The generic components of a shell : the knowledge acquisition, the knowledge Base, the reasoning, the explanation and the user interface are shown below. The knowledge base and reasoning engine are the core components.



1. Knowledge Base

A store of factual and heuristic knowledge. Expert system tool provides one or more knowledge representation schemes for expressing knowledge about the application domain. Some tools use both Frames (objects) and IF-THEN rules. In PROLOG the knowledge is represented as logical statements.

2. Reasoning Engine

Inference mechanisms for manipulating the symbolic information and knowledge in the knowledge base form a line of reasoning in solving a problem. The inference mechanism can range from simple modus ponens backward chaining of IF-THEN rules to Case-Based reasoning.

3. Knowledge Acquisition subsystem

A subsystem to help experts in build knowledge bases. However, collecting knowledge, needed to solve problems and build the knowledge base, is the biggest bottleneck in building expert systems.

4. Explanation subsystem

A subsystem that explains the system's actions. The explanation can range from how the final or intermediate solutions were arrived at justifying the need for additional data.

5. User Interface

A means of communication with the user. The user interface is generally not a part of the expert system technology. It was not given much attention in the past. However, the user interface can make a critical difference in the perceived utility of an Expert system.

6. . Explanation

Most expert systems have explanation facilities that allow the user to ask questions - why and how it reached some conclusion.

The questions are answered by referring to the system goals, the rules being used, and existing problem solving. The rules typically reflect empirical, or "compiled" knowledge. They are codes of an expert's rules of thumb, not the expert's deeper understanding.

Example :

Dialog with an expert system designed to give advice on car problems.

System Does the car start ?

User No.

System Does the engine turn over ?

User Yes.

System Do you smell gas ?

User Yes

System Recommendation - Wait 5 minutes and try again.

User Why ?

System I used the rule:

If not start, and engine_turn_over, and smell_gas

Then recommendation is 'Wait 5 minutes and try again'

User

Note : The rule gives the correct advice for a flooded car, and knows the questions to be ask to determine if the car is flooded, but it does not contain the knowledge of **what** a flooded car is and **why** waiting will help.

Types of Explanation

There are four types of explanations commonly used in expert systems.

1. Rule traces reports on the progress of a consultation;
2. Explanation of how the system reached to the given conclusion;
3. Explanation of why the system did not give any conclusion.
4.Explanation of why the system is asking a question.

6. What is knowledge acquisition? Explain in detail. (November 2013) (MAY/JUNE 2016)

Knowledge acquisition includes the elicitation, collection, analysis, modeling and validation of knowledge.

Issues in Knowledge Acquisition

The important issues in knowledge acquisition are:

- knowledge is in the head of experts
- Experts have vast amounts of knowledge
- Experts have a lot of tacit knowledge
  - They do not know all that they know and use
  - Tacit knowledge is hard (impossible) to describe
- Experts are very busy and valuable people
- One expert does not know everything
- Knowledge has a "shelf life"

Techniques for Knowledge Acquisition

The techniques for acquiring, analyzing and modeling knowledge are :

1. Protocol-generation techniques.
2. Protocol analysis techniques.
3. Hierarchy generation techniques.
4. Matrix-based techniques.
5. Sorting techniques.
6. Limited-information and constrained-processing tasks.
7. Diagram-based techniques.

■ Protocol-generation techniques

Include many types of interviews (unstructured, semi-structured and structured), reporting and observational techniques.

■ **Protocol analysis techniques**

Used with transcripts of interviews or text-based information to identify basic knowledge objects within a protocol, such as goals, decisions, relationships and attributes. These act as a bridge between the use of protocol-based techniques and knowledge modeling techniques.

Hierarchy-generation techniques

Involve creation, reviewing and modification of hierarchical knowledge. Hierarchy-generation techniques, such as laddering, are used to build taxonomies or other hierarchical structures such as goal trees and decision networks. The Ladders are of various forms like concept ladder, attribute ladder, composition ladders**.**

Matrix-based techniques

Involve the construction and filling-in a 2-D matrix (grid, table), indicating such things, as may be, for example, between concepts and properties (attributes and values) or between problems and solutions or between tasks and resources, etc. The elements within the matrix can contain: symbols (ticks, crosses, question marks) , colors , numbers , text.

Sorting techniques

Used for capturing the way people compare and order concepts; it **may reveal knowledge about classes, properties and priorities.**

Limited-information and constrained-processing tasks

Techniques that either limit the time and/or information available to the expert when performing tasks. For example, a twenty-questions technique provides an efficient way of accessing the key information in a domain in a prioritized order.

Diagram-based techniques

Include generation and use of concept maps, state transition networks, event diagrams and process maps. These are particularly important in capturing the "what, how, when, who and why" of tasks and events.

7. Explain the characteristics, role and advantages of expert system? (NOV 2013) (MAY/JUNE 2016)

Expert system operates as an interactive system that responds to questions, asks for clarifications, makes recommendations and generally aids the decision-making process.

Expert systems have many Characteristics:

1. Operates as an interactive system

This means an expert system:

- Responds to questions
- Asks for clarifications
- Makes recommendations
- Aids the decision-making process.

2. Tools have ability to sift (filter) knowledge

- Storage and retrieval of knowledge
- Mechanisms to expand and update knowledge base on a continuing basis.

3. Make logical inferences based on knowledge stored

- Simple reasoning mechanisms is used.
- Knowledge base must have means of exploiting the knowledge stored, else it is useless; e.g., learning all the words in a language, without knowing how to combine those words to form a meaningful sentence.

4. Ability to Explain Reasoning

- Remembers logical chain of reasoning; therefore user may ask
    - for explanation of a recommendation
    - factors considered in recommendation
- Enhances user confidence in recommendation and acceptance of expert system

5. Domain-Specific

- A particular system caters a narrow area of specialization; e.g., a medical expert system cannot be used to find faults in an electrical circuit.
- Quality of advice offered by an expert system is dependent on the amount of knowledge stored.

6. Capability to assign Confidence Values

- Can deliver quantitative information
- Can interpret qualitatively derived values
- Can address imprecise and incomplete data through assignment of confidence values.

7. Applications

- Best suited for those dealing with expert heuristics for solving problems.
- Not a suitable choice for those problems that can be solved using purely numerical techniques.

8. Cost-Effective alternative to Human Expert

- Expert systems have become increasingly popular because of their specialization, albeit in a narrow field.
- Encoding and storing the domain-specific knowledge is economic process due to small size.
- Specialists in many areas are rare and the cost of consulting them is high; an expert system of those areas can be useful and cost-effective alternative in the long run.

ROLE OF EXPERT SYSTEMS

The Expert systems have found their way into most areas of knowledge work. The applications of expert systems technology have widely proliferated to industrial and commercial problems, and even helping NASA to plan the maintenance of a space shuttle for its next flight. The main applications are

1. Diagnosis and Troubleshooting of Devices and Systems

Medical diagnosis was one of the first knowledge areas to which Expert system technology was applied in 1976. However, the diagnosis of engineering systems quickly surpassed medical diagnosis.

2. Planning and Scheduling

The Expert system's commercial potential in planning and scheduling has been recognized as very large. Examples are airlines scheduling their flights, personnel, and gates; the manufacturing process planning and job scheduling;

3. Configuration of Manufactured Objects from sub-assemblies

Configuration problems are synthesized from a given set of elements related by a set of constraints. The Expert systems have been very useful to find solutions. For

example, modular home building and manufacturing involving complex engineering design.

4. Financial Decision Making

The financial services are the vigorous user of expert system techniques. Advisory programs have been created to assist bankers in determining whether to make loans to businesses and individuals. Insurance companies to assess the risk presented by the customer and to determine a price for the insurance. ES are used in typical applications in the financial markets / foreign exchange trading.

5. Knowledge Publishing

This is relatively new, but also potentially explosive area. Here the primary function of the Expert system is to deliver knowledge that is relevant to the user's problem. The two most widely known Expert systems are : one, an advisor on appropriate grammatical usage in a text; and the other, is a tax advisor on tax strategy, tactics, and individual tax policy.

6. Process Monitoring and Control

Here Expert system does analysis of real-time data from physical devices, looking for anomalies, predicting trends, controlling optimality and failure correction. Examples of real-time systems that actively monitor processes are found in the steel making and oil refining industries.

7. Design and Manufacturing

Here the Expert systems assist in the design of physical devices and processes, ranging from high-level conceptual design of abstract entities all the way to factory floor configuration of manufacturing processes.

ES usage provides many advantages. Some of the advantages are:

(a) Consistency

One of the advantages of an ES is that the results given are consistent. This might be due to the fact that there are no elements such as exhaustion and emotions as experienced by humans.

(b) Hazardous Working Environment

Through an ES, we can avoid exposing ourselves to a toxic or radioactive environment. An ES can take over the place of an expert to handle problems in a high-risk area such as a nuclear power plant.

(c) Ability to Solve Complex and Difficult Problems

A very difficult problem encountered by an organisation, if not taken seriously, can cause an adverse impact such as losses or cancellation of a business deal. Sometimes, the problems need to be attended to quickly. The problems can become more complicated when individuals or experts involved in solving them are absent or cannot be contacted. Thus, an ES serves as an alternative to experts.

(d) Combination of Knowledge and Expertise from Various Sources

One of the important components in an ES is the knowledge base. This component contains the accumulated knowledge and acquired or transferred expertise from many experts. Thus, an ES is sometimes more superior than an expert because its knowledge and expertise have come from many sources.

(e) Training Tool for Trainees

An ES can be used by trainees to learn about the knowledge-based system. Trainee who uses an ES would be able to observe how an expert solves a problem.

Several weaknesses concerning the use of ES.

(a) Not Widely Used

ES is not widely used in business firms or organisations. Due to limited usage, firms are still in doubt about the capability and, most definitely, the high cost involved in investing in an ES.

(b) Difficult to Use

Using an ES is very difficult and learning and mastering it requires a long time. This discourages managers from using ES. In one aspect, developing an ES that is user-friendly is the biggest challenge for ES developer.

(c) Limited Scope

This is the most obvious weakness in an ES; its scope is very limited to its field only. In the development aspect, the ES built is best developed because of its high accuracy. However, usage-wise decision makers face constantly changing problems which involve different fields that are interrelated.

(d) Probable Decision Error

The main source of the knowledge is experts. Humans make mistakes. If the experts input wrong information into the Expert system, this will have a negative impact on the results produced.

(e) Difficult to Maintain

The information in ES must be constantly updated to solve new problems. Every new problem that occurs needs new knowledge and expertise. This means that there must be an on-going relationship between the domain experts and the ES developer. This situation requires the domain experts update the source of knowledge and expertise to suit the current situation.

(f) Costly Development

The cost to consult a group of experts is not cheap, what if ES was built traditionally without involving the use of an Expert System shell? On the other hand, programming cost is high because the artificial intelligence technique is difficult to master and needs a very skilful programmer.

(g) Legal and Ethical Dilemma

We must be responsible for our actions and decisions. An expert has to take responsibility for the information he or she provides. . The difficult question here is who should shoulder the responsibility if a decision suggested by ES results in a negative outcome.

Industry/ Practical Connectivity

Industry Connectivity

- The student can work as a software engineer in industry, which is working for companies like Amazon to shopping list recommendation engines or Facebook analyzing and processing big data.
- The students can also work as a hardware engineer developing electronic parking assistants or home assistant robots.

Latest developments in AI

- It can be used in biological research such as detection of diseases.
- The Stanford university team builds Shazam, an expert system for detecting earthquakes.
- Gabriel is A New Artificial Intelligence Named after the Messenger Angel that acts as a personal cognitive assistant and whispers instructions into human's ear.

Question Paper Code : 21312

B.E./B.Tech. DEGREE EXAMINATION, MAY/JUNE 2013.

Sixth Semester

Computer Science and Engineering

CS 2351/CS 61/10144 CS 601 — ARTIFICIAL INTELLIGENCE

(Common to Seventh Semester – Electronics and Instrumentation Engineering)

(Regulation 2008 /2010)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. What are the four components to define a problem? Define them.
2. Define basic agent programs.
3. Define universal and existential quantifiers.
4. What is skolemization?
5. What is a consistent plan?
6. Define critical path.
7. State Baye's rule.
8. Give the full specification of a Bayesian network.
9. Distinguish between supervised and unsupervised learning.
10. Define support vectors.

PART B — (5 × 16 = 80 marks)

11. (a) (i) What is uninformed search? Explain depth first search with example. (8)

(ii) Give the algorithm for recursive best first search. (8)

Or

(b) (i) Explain the nature of heuristics with an example. What is the effect of heuristic accuracy on performance? (8)

(ii) Write a simple back tracking algorithm for constraint satisfaction problems. (8)

12. (a) (i) Explain the steps involved in knowledge engineering projects with example. (10)

(ii) Give the five logical connectives used to construct complex sentences and give the formal grammar of propositional logic. (6)

Or

(b) (i) Discuss backward chaining algorithm. (8)

(ii) Explain the algorithm for computing more general unifiers. (8)

13. (a) (i) Discuss state space search based planning algorithms. (8)

(ii) Explain GRAPHPLAN algorithm in detail. (8)

Or

(b) (i) What way will you modify POP to incorporate HTN planning? Explain with example. (8)

(ii) Discuss various planning methods for handling indeterminacy. (8)

14. (a) (i) Write the enumeration algorithm for answering queries on Bayesian networks. (8)

(ii) Describe a method for constructing Bayesian networks. (8)

Or

(b) (i) Discuss smoothing with equations and algorithm. (8)

(ii) Explain the basic concepts of hidden Markov model and its role in smoothing. (8)

15. (a) (i) Give decision tree learning algorithm and explain. (8)

(ii) Explain EM algorithm. (8)

Or

(b) (i) Explain back propagation process with its algorithm. (8)

(ii) What is passive ADP agent? Give full agent program for a passive ADP agent. (8)

Reg. No. : | | | | | | | | | | | |

Question Paper Code : 31312

B.E./B.Tech. DEGREE EXAMINATION, NOVEMBER/DECEMBER 2013.

Sixth Semester

Computer Science and Engineering

CS 2351/CS 61/10144 CS 601 – ARTIFICIAL INTELLIGENCE

(Common to Seventh Semester – Electronics and Instrumentation Engineering)

(Regulation 2008/2010)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. What are software agents?
2. Define the effect of heuristic accuracy on performance.
3. Define the first order definite clause.
4. State the expressiveness extension.
5. Define the bi-directed search.
6. What are continuous random variables?
7. What is partial order Planning?
8. Define temporal models.
9. State the types of approximation methods.
10. Define entailment constraints.

PART B — (5 × 16 = 80 marks)

11. (a) What are the five uninformed search strategies? Explain any two in detail with example.

Or

(b) Explain the approach of formulation for constraint satisfaction problems with example.

12. (a) Explain the forward chaining process and efficient forward chaining with example. State its usage.

Or

(b) State and explain the various steps in knowledge engineering process.

13. (a) Explain the procedure of planning with state space search with example.

Or

(b) Explain the process of scheduling with resource constraints in detail with suitable example.

14. (a) (i) How to handle uncertain knowledge with example? (8)

(ii) How to represent knowledge in an uncertain domain. (8)

Or

(b) (i) Explain the hidden markov model. (8)

(ii) Explain the need of fuzzy set and fuzzy logic with example. (8)

15. (a) (i) Explain the process of learning on action utility function. (8)

(ii) Explain the temporal difference learning with example. (8)

Or

(b) What are various approaches for instance based learning. Explain any one with example.

Question Paper Code : 51352

B.E./B.Tech. DEGREE EXAMINATION, MAY/JUNE 2014.

Sixth Semester

Computer Science and Engineering

CS 2351/ CS 61— ARTIFICIAL INTELLIGENCE

(Common to Seventh Semester –Electronics and Instrumentation Engineering)

(Regulation 2008/2010)

(Common to PTCS 2351 –Artificial Intelligence for B.E (Part–Time) Sixth Semester Computer Science and Engineering – Regulation 2009)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — ($10 \times 2 = 20$ marks)

1. Give the structure of an agent in an environment.

2. List the criteria to measure the performance of search strategies.

3. $P \Rightarrow Q \Rightarrow \neg PVQ$ Construct a truth table to show that this equivalence holds.

4. Write the generalized modus ponens rule.

5. List out the various planning techniques.

6. What is contingency planning?

7. List down two applications of temporal probabilistic models.

8. Define uncertainty.

9. List some applications where reinforcement learning is used.

10. What are the three factors involved in the analysis of efficiency gains from EBL(Explanation Based Learning)?

PART B — (5 × 16 = 80 marks)

11. (a) (i) What are the problems caused due to incomplete knowledge on the states or actions? Define each with example. (8)

(ii) Describe constraint satisfaction problem in detail. (8)

Or

(b) (i) Explain the components of problem definition with an example. (8)

(ii) Briefly explain the search strategies in uninformed search. (8)

12. (a) (i) What are the steps to convert First order logic sentence to Normal form? Explain each step. (6)

(ii) Represent the following sentences in predicate logic and convert the following sentences to CNF form

(1) All women who like ice-creams like chocolates. (2)

(2) No man is happy with a spendthrift wife. (2)

(3) The best movie in Hollywood is always better than the best movie in Bollywood. (2)

(4) Some people like eating outside all the time and some people like eating at home all the time (2)

(5) It might be argued that one aspect of intelligent behavior is the ability to infer new facts about the world by combining existing ones. Has the theory of logic given us a tool to allow computers to display this sort of intelligence? Can humans make other leaps of inference that are impossible with logic alone? (2)

Or

(b) (i) Differentiate propositional logic with FOL. List the inference rules along with suitable examples for First order Logic. (10)

(ii) Consider the following sentences:

(1) John likes all kinds of food.

(2) Apples are food.

(3) Chicken is food.

(4) Anything anyone eats and isn't killed alive.

(5) Sue eats everything bill eats.

(A) Translate these sentences into formulas in predicate logic.

(B) Convert the formulas of a part into clause form.

(C) Prove that "John likes peanuts" using forward chaining.

(D) Prove that "John likes peanuts" using backward chaining.

15. (a) (i) Suppose you set this as a machine learning problem to a learning agent. You specify that the positives are the exams which are difficult. (10)

(1) Which would the most specific hypothesis that the agent would initially construct?

(2) How would the agent generalize this hypothesis in light of the second positive example? What are the other possible generalizations are there?

(3) How would you use the 2 hypotheses (from parts (1) and (2) to predict whether an exam will be difficult?

(4) What score would the 2 hypotheses get in terms of predictive accuracy over the training set, and which one would be chosen as the learned hypothesis?

(ii) Consider the problem of learning to play tennis. Are there aspects of this learning that are supervised learning? Is this supervised learning or reinforcement learning? (6)

Or

(b) (i) Consider a simple domain: waiting at a traffic light. Give an example of a decision tree for this domain. (8)

(1) Make a list of relevant variables.

(2) Explain how we can use the concept of information or expected information gain to determine which variable to choose, for a maximally compact decision tree.

(ii) For the case of learning to play tennis (or some other sport with which you are familiar). Is this supervised learning or reinforcement learning? (8)

Reg. No. : | | | | | | | | | | | | |

Question Paper Code : 71390

B.E./B.Tech. DEGREE EXAMINATION, APRIL/MAY 2015.

Sixth Semester

Computer Science and Engineering

CS 2351/CS 61/10144 CS 601 – ARTIFICIAL INTELLIGENCE

(Common to Seventh Semester – Electronics and Instrumentation Engineering/Instrumentation and Control Engineering/Information Technology)

(Regulation 2008/2010)

(Common to PTCS 2351/10144 CS 601 – Artificial Intelligence for B.E. (Part-Time) Sixth Semester – Computer Science and Engineering – Regulation 2009/2010)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. Define Ideal Rational agents

2. Why Problem formulation must follow goal Formulation?

3. Differentiate Forward Chaining and Backward Chaining.

4. What is the use of Online search agent in unknown environment

5. Define Partial order Planner

6. What are the differences and similarity in problem solver and planner?

7. List down the applications of Bayesian network.

8. Define Uncertainty. How it is solved?

9. What are the methods of Statistical learning?

10. State the advantages of Inductive learning.

PART B — (5 × 16 = 80 marks)

11. (a) (i) Explain any two Informed Search Strategies. (10)

(ii) Discuss about Constraint satisfaction Problem. (6)

Or

(b) Explain the following uninformed search Strategies

(i) Depth first Search (6)

(ii) Iterative Deepening Depth First Search. (6)

(iii) Bidirectional Search. (4)

12. (a) Explain Forward chaining and Backward chaining algorithm with an example.

Or

(b) (i) Illustrate the use of First Order Logic to represent Knowledge. (10)

(ii) Write Short note on Unification (6)

13. (a) Explain the concept of planning with state space search using suitable examples. (16)

Or

(b) Explain the use of planning graph in providing better heuristic estimation with suitable example. (16)

14. (a) (i) Explain the Inference in Temporal models. (10)

(ii) Write short notes on Hidden Markov model. (6)

Or

(b) Explain about the exact inference in Bayesian networks. (16)

15. (a) The following table consists of training data from an employee database. The data have been generalized. Let status be the class label attribute. Construct Decision tree from the given data

Department	Age	Salary	Count	Status
Sales	31...35	46K...50K	30	Senior
Sales	26...30	26K...30K	40	Junior
Sales	31...35	31K...35K	40	Junior
Systems	21...25	46K...50K	20	Junior
Systems	31...35	66K...70K	5	Senior
Systems	26...30	46K...50K	3	Junior
Systems	41...45	66K...70K	3	Senior
Marketing	36...40	46K...50K	10	Senior
Marketing	31...35	41K...45K	4	Junior
Secretary	46...50	36K...40K	4	Senior
Secretary	26...30	26K...30K	6	Junior

Or

(b) Explain in detail about Active and Passive Reinforcement learning. (16)

Question Paper Code : 57262

B.E./B.Tech. DEGREE EXAMINATION, MAY/JUNE 2016

Sixth Semester

Computer Science and Engineering

CS 6659 – ARTIFICIAL INTELLIGENCE

(Common to fifth semester Instrumentation and Control Engineering and Electronics and Instrumentation Engineering and Sixth Semester Information Technology)

(Regulations 2013)

Time : Three Hours **Maximum : 100 Marks**

Answer ALL questions.

PART – A (10 × 2 = 20 Marks)

1. What is ridge ? [Page No:8]
2. How much knowledge would be required by a perfect program for the problem of playing chess ? Assume that unlimited computing power is available. [Page No:9]
3. What is alpha-beta pruning ? [Page No:44]
4. For the given sentence "All Pompeians were Romans" write a well formed formula in predicate logic. [Page No:47]
5. What is Bayesian Networks ? [Page No:69]
6. Write the properties of fuzzy sets. [Page No:68]
7. What is rote learning ? [Page No:90]
8. Brief frame problem. [Page No:90]
9. What is meta knowledge ? How meta knowledge is represented in rule-based expert systems ? [Page No:124]
10. Write any four earliest expert systems. [Page No:124]

1 57262

PART – B (5 × 16 = 80 Marks)

11. (a) (i) Explain the Heuristic functions with examples. [Page No:9] **(6)**

(ii) Write the algorithm for Generate and Test and simple Hill Climbing. [Page No:34] **(10)**

OR

(b) Solve the given problem. Describe the operators involved in it. **(16)**

Consider 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 ? Explicit Assumptions: A jug can be filled from the pump, water can be poured out of a jug onto the ground, water can be poured from one jug to another and that there are no other measuring devices available. [Page No:31]

12. (a) Convert the following well formed formula into clause form with sequence of steps. **(16)**

$\forall x$: [Roman(x) $\wedge$ Know(x,Marcus)] $\rightarrow$ [hate(x, Caesar) $\vee$ ($\forall y$: $\exists z$: hate(y,z) $\rightarrow$ thinkcrazy(x,y))] [Page No:62]

OR

(b) (i) Write the resolution procedure for prepositional logic. [Page No:50] **(8)**

(ii) Explain the Iterative Deepening Algorithm. [Page No:45] **(8)**

13. (a) (i) Briefly explain how reasoning is done using fuzzy logic. [Page No:79] **(6)**

(ii) Explain Dempster-Shafer Theory. [Page No:86] **(10)**

OR

(b) What is Forward Chaining and how does it work ? Explain the forward Chaining algorithm with an example. [Page No:70]. **(16)**

14. (a) (i) Describe the components of a planning system. [Page No:92] **(10)**

(ii) What is ID3 ? Write the drawback of ID3. [Page No:122] **(6)**

OR

(b) (i) Describe the Hierarchical planning method with an example. [Page No:91] **(8)**

(ii) Describe the Learning with Macro-Operators. [Page No:98] **(8)**

15. (a) (i) Explain about the Knowledge acquisition. [Page No:145] **(10)**

(ii) Write the characteristic features of Expert systems. [Page No:143] **(6)**

OR

(b) (i) Explain the basic components of an expert system. [Page No:127] **(10)**

(ii) Write any six applications of expert systems. [Page No:143] **(6)**
