

INTRODUCTION TO AI AND INTELLIGENT AGENT

PREVIOUS YEARS QUESTIONS

PART-A

Prob.1 Explain Artificial Intelligence in brief.

Sol. According to the father of AI, John McCarthy it is "The science and Engineering of making intelligent machines, especially intelligent computer programs".

Prob.2 What is the goal of Artificial Intelligence?

Sol.(a) To Create Expert System: It is the type of system in which the system exhibit intelligent behaviour, and advice to its users.

(b) To Implement Human Intelligence in Machines: It is a way of creating the systems that understand, think, learn and behave like humans.

Prob.3 Explain types of Artificial Intelligence.

Sol. Strong Artificial Intelligence: It deals with the creation of real intelligence artificially. Also, strong AI believes that machines can be made sentient.

Weak Artificial Intelligence: It doesn't believe creating human level intelligence in machines is possible.

Prob.4 What is Constraint Satisfaction Problems?

Sol. Constraint Satisfaction Problems (CSPs) are mathematical problems defined as a set of objects, the state of which must meet several constraints. CSPs are useful for AI because the regularity of their formulation offers commonality for analyzing and solving problems.

Prob.5 What are the various applications of Artificial Intelligence?

Sol.

- Natural Language Processing
- Expert Systems
- Gaming
- Vision Systems
- Intelligent Robots
- Speech Recognition etc.

PART-B

Prob.6 What are the major categories of AI? Explain them briefly. Why AI is a matter of research ?

[R.T.U. 2012]

Sol. Artificial Intelligence: AI system has four major categories

- (1) Knowledge Representation
- (2) Heuristic Search
- (3) AI Programming Language and Tools
- (4) AI Hardware

(1) The quality of the result depends on how much knowledge the system possesses. The available knowledge must be represented in a very efficient way. Hence, knowledge representation is a vital component of the system.

(2) It is not merely enough that knowledge is represented efficiently. The inference process should also be equally good for satisfactory results. The inference process is broadly divided into brute and heuristic search procedures.

Uniform Cost Search: Generally, in the search method, with every edge that we traverse, there is a cost associated with it. With BFS, we are assuming that every edge is having the same cost. Now speaking about the uniform cost search, if all the edges do not have the same cost, the breadth first search generalises to uniform cost search. So, the expansion takes place in the order of costs of the edges from the root rather than the order of the depth.

At every step, the next thing to be performed expands/ selects a node X , whose cost $c(X)$ is lowest, where $c(X)$ is the sum of cost of edges from root to the node. It is the core step that is followed while performing the uniform cost search. This search technique can be easily implemented using a priority queue.

It is quiet obvious that the search is least concerned about the steps it needs to carry out to reach the goal step. The only concern is the cost. So, there could be a possibility that the search could get in a loop. The search is said to ensure completeness if the cost at every edge is greater or equal to some constant. It, therefore, satisfies the optimal property as well. The uniform search has $O(b^d)$ space and time complexities, as it mostly explores large trees.

Let us consider an example to show how it works.

Consider the following tree given in Fig., where the uniform cost search is to be applied from the start, i.e., node A to the goal node G. We will use a frontier and an expanded list while generating the solution. (An explored list can also be maintained to keep track of the explored nodes. This list is not considered in this example to avoid confusion.)

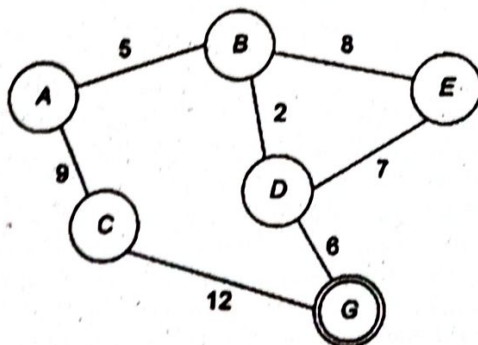


Fig. Input tree for uniform cost search.

Let us discuss how the uniform cost search proceeds step-wise.

Step 1: Start from A

Expanded none

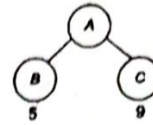
Step 2: Frontier

Expanded none



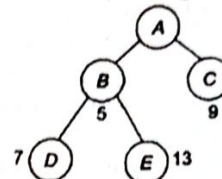
Step 3: Frontier: B and C

Expanded A



Select the lowest, i.e., B

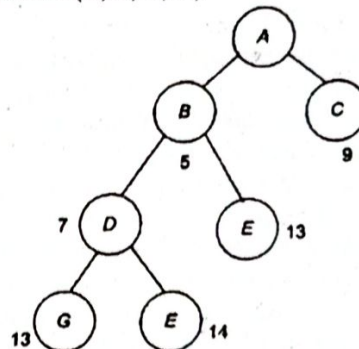
Step 4: Frontier: D, C, E (priority queue) Expanded A, B



Note: Remember the costs are added from the root till the node in consideration. Select the lowest, i.e., D.

Step 5: Frontier (C, E, E, G)

Expanded A, B, D



In step 5, we have considered E, as its parent is different. Since we have reached the goal state, from the tree, we can see that we have found a path. The path is A-B-D-G. With the frontiers and the tree generation process, we can easily have the solution to the problem in uninformed search. Dijkstra's approach is often considered to be variant of this uniform cost search.

Comparison of the Uninformed Techniques: Table summarizes the complexities, optimality and completeness about the search techniques.

Table: Comparison of Uninformed Search Techniques

Search Techniques	Complete	Optimal	Time complexity	Space complexity
Breadth first search	Yes	Yes	$O(b^d)$	$O(b^d)$
Uniform cost	Yes	Yes	$O(b^d)$	$O(b^d)$
Depth first	No	No	$O(b^d)$	$O(bd)$
Depth limited	No	No	$O(b^l)$	$O(bl)$
Iterative deepening	Yes	Yes	$O(b^d)$	$O(bd)$
Bi-directional	Yes	Yes	$O(b^{d/2})$	$O(b^{d/2})$

Prob.15 *What is AI and AI techniques? Briefly explain how AI technique can be represented. List out some of task domain of AI?*
[R.T.U. 2019, 2015]

Sol. Artificial Intelligence : Artificial Intelligence (AI) is usually defined as the science of making computers do things that require intelligence when done by humans. AI has had some success in limited, or simplified, domains. However, the five decades since the inception of AI have brought only very slow progress, and early optimism concerning the attainment of human-level intelligence has given way to an appreciation of the profound difficulty of the problem. Research associated with artificial intelligence is highly technical and specialized. The core problems of artificial intelligence include programming computers for certain traits such as:

- Knowledge
- Reasoning
- Problem solving
- Perception
- Learning
- Planning
- Ability to manipulate and move objects

AI Techniques

- (i) Perception
 - (a) Machine vision
 - (b) Speech understanding
 - (c) Touch (tactile or haptic) sensation

- (ii) Robotics
- Natural Language Processing
 - Natural Language Understanding
 - Speech Generation
 - Language Translation
- (iv) Planning
- Expert Systems
 - Machine Learning
 - Theorem Proving
 - Symbolic-Mathematics
- (ix) Game Playing
- (i) Perception
- Machine Vision** : It is easy to interface a TV camera to a computer and get an image into memory; the problem is understanding what the image represents. Vision takes lots of computation; in humans, roughly 10% of all calories consumed are burned in vision computation.
 - Speech Understanding** : Speech understanding is available now. Some systems must be trained for the individual user and require pauses between words. Understanding continuous speech with a larger vocabulary is harder.
 - Touch (Tactile or haptic) Sensation**: Important for robot assembly tasks.
 - Robotics**
 - Although industrial robots have been expensive, robot hardware can be cheap. Radio Shack has sold a working robot arm and hand for \$15. The limiting factor in application of robotics is not the cost of the robot hardware itself.
 - What is needed is perception and intelligence to tell the robot what to do. "blind" robots are limited to very well-structured tasks (like spray painting car bodies).
 - Natural Language Understanding** : Natural languages are human languages such as English. Making computers understand English allows non-programmers to use them with little training. Applications in limited areas (such as access to data bases) are easy.
 - Natural Language Generation** : Easier than NL understanding. Can be an inexpensive output device.
 - Machine Translation** : Usable translation of text is available now. Important for organizations that operate in many countries. In a not too far future develops for eleven-year old David in a research lab the first intelligent robot with

human feelings in the shape. But its "foster parents" are overtaxed with the artificial spare child and suspend it. Posed on itself alone David tries to fathom its origin and the secret of its existence.

(iv) Planning

- Planning attempts to order actions to achieve goals.
- Planning applications include logistics, manufacturing scheduling, planning manufacturing steps to construct a desired product.
- There are huge amounts of money to be saved through better planning.

(v) Expert Systems

Expert systems attempt to capture the knowledge of a human expert and make it available through a computer program. There have been many successful and economically valuable applications of expert systems.

Benefits

- Reducing skill level needed to operate complex devices.
- Diagnostic advice for device repair.
- Interpretation of complex data.
- "Cloning" of scarce expertise.
- Capturing knowledge of expert who is about to retire.
- Combining knowledge of multiple experts.
- Intelligent training.

(vi) Theorem Proving

Proving mathematical theorems might seem to be mainly of academic interest. However, many practical problems can be cast in terms of theorems. A general theorem prover can therefore be widely applicable.

Examples

- Automatic construction of compiler code generators from a description of a CPU's instruction set.
- J Moore and colleagues proved correctness of the floating-point division algorithm on AMD CPU chip.
- Symbolic Mathematics**
Symbolic mathematics refers to manipulation of formulas, rather than arithmetic on numeric values.

(a) Algebra

(b) Differential and Integral Calculus

Symbolic manipulation is often used in conjunction with ordinary scientific computation as a generator of programs used to actually do the calculations. Symbolic manipulation programs are an important component of scientific and engineering workstations.

(viii) Game Playing

Games are good vehicles for research because they are well formalized, small, and self-contained. They are therefore easily programmed.

Games can be good models of competitive situations, so principles discovered in game-playing programs may be applicable to practical problems.

Task Domains of AI

Task domains of AI is numerous and ever increasing. Following are most common applications.

(i) **Game Playing**: One can buy machines that can play master level chess for a few hundred dollars. There is some AI in them, but they play well against people mainly through brute force computation looking at hundreds of thousands of positions.

(ii) **Speech Recognition**: In the 1990s, computer speech recognition reached a practical level for limited purposes. Thus United Airlines has replaced its keyboard tree for flight information by a system using speech recognition of flight numbers and city names.

(iii) **Understanding Natural Language**: Just getting a sequence of words into a computer is not enough. The computer has to be provided with an understanding of the domain the text is about, and this is presently possible only for very limited domains.

(iv) **Computer Vision**: The world is composed of three-dimensional objects, but the inputs to the human eye and computers TV cameras are two dimensional. Some useful programs can work solely in two dimensions, but full computer vision requires partial three-dimensional information that is not just a set of two-dimensional views. At present there are only limited ways of representing three-dimensional information directly, and they are not as good as what humans evidently use.

(v) **Expert Systems (ES)**: A computer program that contains a knowledge base and a set of algorithms or rules that infer new facts from knowledge and from incoming data. An expert system is an artificial intelligence application that uses a knowledge base of human expertise to aid in solving problems. The degree of problem solving is based on the quality of the data and rules obtained from the human expert. Expert systems are designed to perform at a human expert level. In practice, they will perform both well below and well above that of an individual expert. The expert system derives its answers by running the knowledge base through an inference engine, a software program that interacts with the user and processes the results from the rules and data in the knowledge base. Expert systems are used in applications such as medical diagnosis, equipment repair, investment analysis, financial, estate and insurance planning, route scheduling for delivery

vehicles, contract bidding, counseling for self-service customers, production control and training.

(vi) **Reasoning of Humans**: Normally people create categories for reasoning. For example, cash is a current asset and a current asset is an asset. Thus they categorize assets. They use specific rules, a priori rules to give priority. Rules can be cascaded like:

"If A then B"
"If B then C"
A--->B--->C

They also use heuristics, "rules of thumb". Heuristics can be captured using rules like "If the meal includes red meat then choose red salad dressing". Heuristics represent conventional wisdom. They use past experience, "cases", particularly evident in precedence-based reasoning e.g. law or choice of accounting principles. Similarity of current case to previous cases provides basis for action choice. Cases are stored using key attributes. Example of attributes can be shown as: cars may be characterized by: year of car.

Prob 16 Discuss comparison between DFS & BFS with various types of control strategies. [R.T.U. 2019, 2014]

OR

Differentiate the breadth first search and depth first search in detail. [R.T.U. 2013]

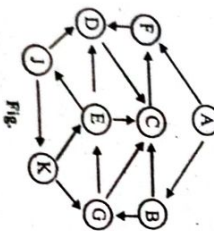
Sol. Control Strategies : Refer to Prob 14.

Both BFS and DFS are graph traversal methods but they differ in following manner:

S. No.	BFS	DFS
(a)	Breadth First Search Works on First In First Out (FIFO) principal.	Depth First Search Works on Last In First Out (LIFO) principal.
(b)	It uses queue as a data structure.	It uses stack as a data structure.
(c)	General Idea: 1. First we examine the starting node A	General Idea: 1. First we examine the starting node A.
	2. Then we examine all the neighbours of A. Then we examine all the neighbour of A, and so on.	2. Then we examine each node N along a path P which begins at A, that is, we process a neighbour of A, then neighbour of A and so on.
	After coming to the end of path P, we backtrack on P until we can continue along another path P, and so on.	

(d) Algorithm	Algorithm
1. Initially, add starting node (say A) to queue	1. Initially, push starting node (say A) onto stack
2. Repeat step 3 & 4 until all nodes visited.	2. Repeat step 3 & 4 until all nodes visited.
3. Delete the front element from queue and add to queue adjacent nodes of deleted nodes which are not visited.	3. Pop the top element from stack and push onto stack all adjacent nodes of popped node which are not visited.
4. Add deleted node into visited list.	4. Add popped element into visited list.
5. Print visited list.	5. Print visited list.
6. END.	6. END.
(e) In BFS, all of the tree that has so far generated must be stored.	DFS requires less memory time only the nodes on the current path are stored.
(f) BFS will not get trapped exploring a blind alley.	DFS may follow a single, unfaithful path for a very long time.
(g) It takes a very long time to reach at its last level	It finds the goal faster.
(h) It requires much memory because it stores all of the child pointers at each level.	It requires less memory because it stores only the pointers of current node.
(i) In BFS, all parts of the tree must be examined to level n before any nodes on level $n+1$ can be examined.	By chance, DFS may find a solution without examining much of the search space at all.
(j) If there is a solution, then BFS is generated to find it. This is generated by the fact that longer paths are never explored until all shorter ones have already been examined.	DFS may find a long path to a solution in one part of the tree, when a shorter path exists in some other, unexplored part of the tree.

Example : Given start node = A



BFS

1. Add A to queue

A visited = { }

2. Delete the front element A from queue and add to queue adjacent nodes of A which are not visited.

B C F visited = {A}

3. Delete the front element B from queue and add to queue adjacent nodes of B which are not visited.

C F G visited = {A, B}

4. Similar to step 2 or 3 we obtain following visited list.

F G visited = {A, B, C}

G D visited = {A, B, C, F}

D E visited = {A, B, C, F, G}

E visited = {A, B, C, F, G, D}

J visited = {A, B, C, F, G, D, E}

K visited = {A, B, C, F, G, D, E, J}

visited = {A, B, C, F, G, D, E, J, K}

i.e. BFS sequence is A, B, C, F, G, D, E, J, K

DFS

1. Push A onto stack

A visited = { }

2. Pop the top element A from stack and push onto stack all adjacent nodes of A which are not visited.

F visited = {A}

C visited = {A}

B visited = {A}

3. Pop the top element F from stack and push onto stack all adjacent nodes of F which are not visited.

D visited = {A, F}

C visited = {A, F}

B visited = {A, F}

4. Similar to step 2 or 3 we obtain following visited list.

C visited = {A, F, D}

B visited = {A, F, D, C}

G visited = {A, F, D, C, B}

E visited = {A, F, D, C, B, G}

J visited = {A, F, D, C, B, G, E}

K visited = {A, F, D, C, B, G, E, J}

visited = {A, F, D, C, B, G, E, J, K}

i.e. DFS sequence is A, F, D, C, B, G, E, J, K

Prob. 17 Write a short note on Hill climbing search strategy.

OR

Explain hill climbing technique.

[R.T.U. 2018]

OR

Discuss and compare hill climbing and Best First search technique? [R.T.U. 2015]

Sol. Hill Climbing / Gradient Descent

Hill climbing algorithms expand the most promising descendant of the most recently expanded node until they encounter the solution. Steepest - ascent hill climbing differs from hill climbing algorithm only the way in which the next node is selected. In this method it selects best successor node for expansion, unlike the first successor node for expansion, as done in hill climbing.

Though this method tries to choose best possible path, but this method, like hill climbing method may fail to find a solution by reaching to a node from where no improvements can be done.

Hill climbing is a mathematical optimization technique which belongs to the family of local search. It is relatively simple to implement, making it a popular first choice. Although

more advanced algorithms may give better results, in some situations hill climbing works just as well.

Hill climbing can be used to solve problems that have many solutions, some of which are better than others. It starts with a random (potentially poor) solution, and iteratively makes small changes to the solution, each time improving it a little. When the algorithm cannot see any improvement anymore, it terminates. Ideally, at that point the current solution is close to optimal, but it is not guaranteed that hill climbing will ever come close to the optimal solution.

For example, hill climbing can be applied to the travelling salesman problem. It is easy to find a solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much better route is obtained.

Hill climbing is used widely in artificial intelligence, for reaching a goal state from a starting node. Choice of next node and starting node can be varied to give a list of related algorithms.

The basic idea of hill climbing is simple: at each current state we select a transition, evaluate the resulting state, and if the resulting state is an improvement we move there, otherwise we try a new transition from where we are.

We repeat this until we reach a goal state, or have no more transitions to try. The transitions explored can be selected at random, or according to some problem specific heuristics.

In some cases, it is possible to define evaluation functions such that we can compute the gradients with respect to the possible transitions, and thus compute which transition direction to take to produce the best improvement in the evaluation function.

Following the evaluation gradients in this way is known as gradient descent.

In neural networks, for example, we can define the total error of the output activations as a function of the connection weights, and compute the gradients of how the error changes as we change the weights. By changing the weights in small steps against those gradients, we systematically minimize the network's output errors.

Hill Climbing

While (3 uphill points):

- Move in the direction of increasing evaluation function f

Let $S_{max} = \arg \max f(s)$, s a successor state to the current state n

- If $f(n) < f(s)$ then move to s
- Otherwise halt at n

Properties

- Terminates when a peak is reached.
- Does not look ahead of the immediate neighbours of the current state.
- Chooses randomly among the set of best successors, if there is more than one.
- Doesn't backtrack, since it doesn't remember where it's been

Best First Search

The idea behind best first search is to always explore the most promising path first can be obtained from the generic search stub function by selecting from L the node with the minimal value of the expected distance to goal (value of heuristic function)

Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.

Judea Pearl described best-first search as estimating the promise of node n by a "heuristic evaluation function $f(n)$ " which, in general, may depend on the description of n , the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain."

Some authors have used "best-first search" to refer specifically to a search with a heuristic that attempts to predict how close the end of a path is to a solution, so that paths which are judged to be closer to a solution are extended first. This specific type of search is called greedy best-first search. Efficient selection of the current best candidate for extension is typically implemented using a priority queue.

Examples of best-first search algorithms include the A^* search algorithm, and in turn, Dijkstra's algorithm (which can be considered a specialization of A^*). Best-first algorithms are often used for path finding in combinatorial search.

Let L be a list of visited but not expanded nodes.

1. Initialize L with the initial state.
2. If L is empty, FAIL, else extract from L node n with minimum value of $h(n)$.
3. If n is a goal node, SUCCEED and return the path from the initial state to n .
4. Remove n from L and insert all the children of n .
5. Goto Step 2.

- Now we have a heuristic, we can use it to direct our search towards the goal.
- Best first search simply chooses the unvisited node with the best heuristic value to visit next.
- It can be implemented in the same algorithm as lowest-cost Breadth First Search.
- This time the priority of each node added to the queue is its heuristic value.

Comparison of Hill Climbing and Best First Search

Hill climbing is a technique in which path is found from starting position to goal node. It finds a better solution by repeatedly changing a single element to the solution. On the other hand best first search finds the best way to find the goal.

S.No.	Hill Climbing	Best First Search
1.	It always flow from current state to its better state.	It scores each state and go towards best score state.
2.	Halt when no successor for current state found.	Move to next node if current node do not provide better solution.
3.	Do not use backtracking.	Uses Backtracking.
4.	Not necessarily find a solution always.	It always found a solution although not an optimal one.
5.	Suffers from Local Maxima Problem.	Do not suffers with Local Maxima.

Prob. 18 What is artificial intelligence? Explain A^* and AO* algorithm in detail. [R.T.U. 2016]

OR

What is artificial intelligence? Explain with suitable example. [R.T.U. 2018]

OR

Explain artificial intelligence with suitable example. [R.T.U. 2017]

Sol. Artificial Intelligence : Refer to Prob. 15.

A^* algorithm : A^* algorithm is specialization of best first search. It provides general guidelines with which to estimate goal distance for general search graphs. At each node along a path to the goal, the A^* algorithm generates all successor nodes and computes an estimates of the chooses the successor with the shortest estimated distance, for expansion. The successors for this node are generated, their distance estimated and the process continue until a goal is found or the search ends in failure. It contain 2 list of node types :

OPEN : Nodes on the open list are nodes that have been generated and had the heuristic function applied to them but which have not been expanded.

CLOSED : Nodes on the closed list are node that have been expanded and whose children are available to the search program.

It also needs three heuristic functions :

f' : It estimates the merits of each node we generate. This will enable to search more promising path first. We call it f' to indicate that it is an approximation to a function f' that gives the true evaluation of the node. For many application it is convenient to define this function as $f' = g + h$

g : The function g is a measure of the cost of getting from the initial state to the current node.

h : The function h is an estimate of the additional cost of getting from the current node to the goal state.

The A^* Algorithm : The best-first search algorithm that was just presented is a simplification of an algorithm called A^* , which was first presented by Hart. This algorithm uses the same f' , g , and h functions, as well as the lists OPEN and CLOSED.

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

- (i) Set SUCCESSOR to point back to BESTNODE. These backwards links will make it possible to recover the path once a solution is found.
- (ii) Compute $g(\text{SUCCESSOR}) = g(\text{BESTNODE}) +$ the cost of getting from BESTNODE to SUCCESSOR.
- (iii) See if SUCCESSOR is the same as any node on OPEN (i.e., it has already been generated but not processed). If so, call that node OLD. Since this node already exists in the graph, we can throw

SUCCESSOR away and add OLD to the list of BESTNODE's successors. Now we must decide whether OLD's parent link should be reset to point to BESTNODE. It should be if the path we have just found to SUCCESSOR is cheaper than the current best path to OLD (since SUCCESSOR and OLD are really the same node). So we see whether it is cheaper to get to OLD via its current parent or to SUCCESSOR via BESTNODE by comparing their g values. If OLD is cheaper (or just as cheap), then we need do nothing. If SUCCESSOR is cheaper, then reset OLD's parent link to point to BESTNODE, record the new cheaper path in $g(\text{OLD})$ and update $f'(\text{OLD})$.

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

algorithm. The first concerns the role of the g function. Thus the A^* algorithm can be used whether we are interested in finding a minimal-cost overall path or simply any path as quickly as possible.

The second observation involves h' , the estimator of h , the distance of a node to the goal. If h' is a perfect estimator of h , then A^* will converge immediately to the goal.

AO* algorithm : AO* algorithm uses a single structure GRAPM, representing the part of the search graph that has been explicitly generated so far. Each node in the graph will point both down to its immediate successors and up to its immediate predecessors. Each node in the graph will also have associated with it an h' value, an estimate of the cost of a path from itself to a set of solution nodes. Also, h' will serve as the estimate of the cost of a path from itself to a set of solution nodes. Also h' will serve as the estimate of goodness of a node. Thus here $f' = h'$ estimates the merit of each nodes we generate.

SOLVED : If any nodes contain a successor are whose descendents are all solved or expanded then node is marked SOLVED. If f' of any node is zero, mark is SOLVED.

FUTILITY : When an unexpanded node is picked up an expanded and if it has no successors then it is assigned the value FUTILITY. This is equivalent to saying that NODE is not solvable.

RESULT, the state that would result if O were applied in O-START.

- (c) If
(FIRST-PART 4- MEA(CURRENT, O-START))
and
(LAST-PART 4- MEM (O-RESULT, GOAL))
are successful, then signal success and return
the result of concatenating
FIRST-PART, O, and LAST-PART*]

Prob.22 Explain the main approach of artificial intelligence?

Sol. Main AI Approaches : AI, as a broad field, encompasses many different approaches ranging from top-down knowledge representation to bottom-up machine learning. There are three related concepts that have been frequently used in recent years: AI, machine learning, and deep learning. In general AI is the broadest concept, machine learning is a sub field in AI, and deep learning is a special type of machine learning. Figure illustrates the relations among these three concepts. While the broad field of AI includes many approaches, its recent popularity is largely due to the outstanding performances of machine learning, especially deep learning. Therefore, this entry focuses on discussing these two types of AI approaches.

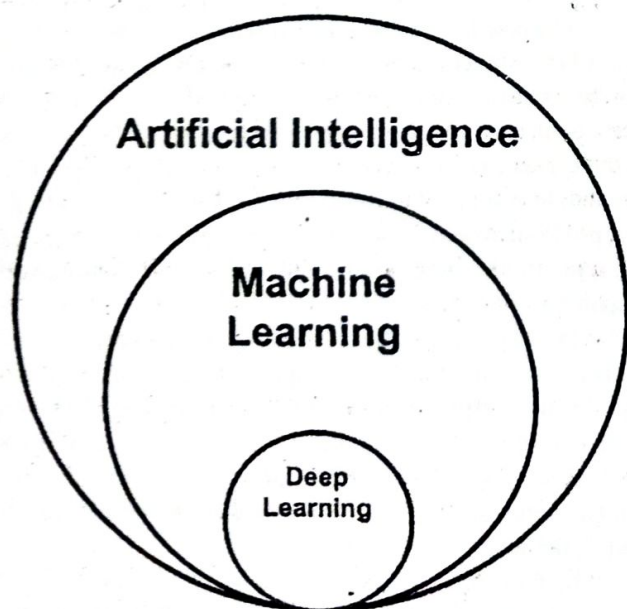


Fig. Relations among AI, machine learning and deep learning

1. Machine learning : Machine learning is a sub field in AI (see Figure) that usually relies on statistical methods or numerical optimization techniques to derive models from data without explicitly programming every model parameter or computing step. One important characteristic shared by many

machine learning models is the use of probability to represent the uncertainty that widely exists in real-world problems. There are three main types of learning: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning requires labeled data for training a computational model while unsupervised learning examines unlabeled data to discover patterns. Reinforcement learning does not require labeled data but needs action-based feedbacks, such as rewards or punishments, to help a computational model to learn. Machine learning tasks can be categorized in various ways. Based on their goals we can identify tasks such as classification, clustering and prediction. In classification, the goal is to classify target into a category e.g., classifying a land parcel into a category such as Commercial or Agricultural. In clustering, the goal is to detect clusters from data e.g. finding the clusters of vehicles based on their locations to detect traffic congestions. In prediction, the goal is to predict unknown values, e.g., predicting the average temperatures of several locations in the near future based on their historical temperatures and other variables using a regression model. There are also other tasks that can fall under the umbrella of machine learning, such as anomaly/ novelty detection, data generation, visualization, feature learning, and others. A variety of machine learning models have been developed, such as regression, decision tree, random forest, support vector machine (SVM), naive Bayesian classifier, density-based clustering, hidden Markov model (HMM), artificial, neural network (ANN), and numerous others. While most machine learning methods can be directly applied to geographic data, they typically do not take into account the uniqueness of geographic phenomena, such as spatial autocorrelation and spatial non-stationarity.

2. Deep learning : Deep learning is a special type of machine learning that focuses on developing and using deep neural networks (DNN) for machine learning tasks DNN is a special type of artificial neural network which has multiple layers (also called hidden layers) between the input and the output layers. Each layer consists of a collection of computing units called neurons, which take the input from the previous layer and generate a non-linear output to the next layer. Deep learning has gained a huge amount of interest in recent years due to its outstanding performances thanks to the availability of large labeled datasets, such as ImageNet and HPC. Similar to other machine learning models, deep learning can be utilized to complete tasks in classification, clustering, prediction, and so forth. Particularly two types of DNN, convolutional neural network (CNN) and recurrent neural network (RNN), have received a lot of attention from the geography community. CNN is especially suitable for processing images by extracting and representing abstract features through a cascade of

neuron layers and using convolutional filters. RNN is suitable for processing sequence data such as movement trajectories (which can be modeled as a sequence of locations) by memorizing some of the previous states and establishing links between the current and previous states. While many studies applied existing models to geographical problems, researchers also developed new DNN models specifically for handling geographic data. Marcos et al (2018) proposed Rotation Equivariant Vector Field Network (RotEqNet) for land cover mapping based on remote sensing images. RotEqNet encodes rotation equivariance in a CNN and can recognize the rotated versions of the same object from remote sensing images while reducing the number of parameters required by a traditional CNN. Srivastava et al (2018) proposed a Variable Input Siamese Convolution Neural Network (VIS-CNN) model for classifying urban object level land use types. Their VIS-CNN model can accept a variable number of Google Street View images for an urban object and aggregate them to learn the land use type in an end-to-end manner.

Prob 23 Explain constraint satisfaction problems in detail.

Sol. Constraint Satisfaction Search: Search can be used to solve problems that are limited by constraints, such as the eight-queens problem. Such problems are often known as Constraint Satisfaction Problems, or CSPs.

In this problem, eight queens must be placed on a chess board in such a way that no two queens are on the same diagonal, row, or column. If we use traditional chess board notation, we mark the columns with letters from a to h and rows with numbers from 1 to 8. So, a square can be referred to by a letter and a number, such as a4 or g7.

This kind of problem is known as a constraint satisfaction problem (CSP) because a solution must be found that satisfies the constraints.

In the case of the eight-queens problem, a search tree can be built that represents the possible positions of queens on the board.

One way to represent this is to have a tree that is 8-ply deep, with a branching factor of 64 for the first level, 63 for the next level, and so on, down to 57 for the eighth level.

A goal node in this tree is one that satisfies the constraints that no two queens can be on the same diagonal, row, or column.

An extremely simplistic approach to solving this problem would be to analyze every possible configuration until one was found that matched the constraints.

A more suitable approach to solving the eight-queens problem would be to use depth-first search on a search tree that represents the problem in the following manner:

B Tech, 07 Sem, C.S. Solved Papers

The first branch from the root node would represent the first choice of a square for a queen. The next branch from these nodes would represent choices of where to place the second queen.

The first level would have a branching factor of 64 because there are 64 possible squares on which to place the first queen.

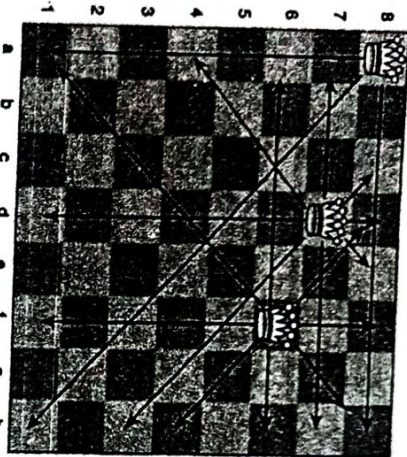


Fig. 1: The eight-queens problem. Three queens have been placed so far.

The next level would have a somewhat lower branching factor because once a queen has been placed, the constraints can be used to determine possible squares upon which the next queen can be placed. The branching factor will decrease as the algorithm searches down the tree. At some point, the tree will terminate because the path being followed will lead to a position where no more queens can be placed on legal squares on the board, and there are still some queens remaining.

In fact, because each row and each column must contain exactly one queen, the branching factor can be significantly reduced by assuming that the first queen must be placed row 1, the second in row 2, and so on. In this way, the first level will have a branching factor of 8 (a choice of eight squares on which the first queen can be placed), the next 7, the next 6, and so on.

In fact, the search tree can be further simplified as each queen placed on the board "uses up" a diagonal, meaning that the branching factor is only 5 or 6 after the first choice has been made, depending on whether the first queen is placed on an edge of the board (columns a or h) or not. The next level will have a branching factor of about 4, and the next may have a branching factor of just 2, as shown in Fig. 1.

Artificial Intelligence

The arrows in Fig. 1 show the squares to which each queen can move. Note that no queen can move to a square that is already occupied by another queen.

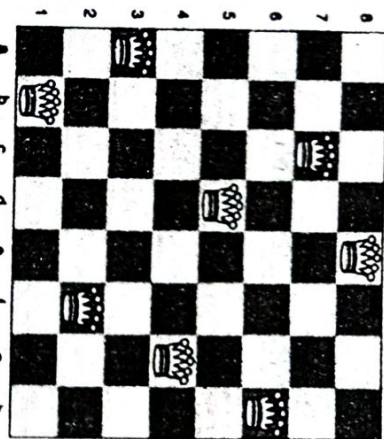


Fig. 2: A solution to the eight-queens problem

In Figure 1, the first queen was placed in column a of row 8, leaving six choices for the next row. The second queen was placed in column d of row 7, leaving four choices for row 6. The third queen was placed in column f in row 6, leaving just two choices (column c or column h) for row 5.

Using knowledge like this about the problem that is being solved can help to significantly reduce the size of the

AI-21

search tree and thus improve the efficiency of the search solution.

A solution will be found when the algorithm reaches depth 8 and successfully places the final queen on a legal square on the board. A goal node would be a path containing eight squares such that no two squares shared a diagonal, row, or column.

One solution to the eight-queens problem is shown in Figure 2.

Note that in this solution, if we start by placing queens on squares e8, c7, h6, and then d5, once the fourth queen has been placed, there are only two choices for placing the fifth queen (b4 or g4). If b4 is chosen, then this leaves no squares that could be chosen for the final three queens to satisfy the constraints. If g4 is chosen for the fifth queen, as has been done in Fig. 2, only one square is available for the sixth queen (a3), and the final two choices are similarly constrained. So, it can be seen that by applying the constraints appropriately, the search tree can be significantly reduced for this problem.

Most-Constrained variables: Using chronological backtracking in solving the eight-queens problem might not be the most efficient way to identify a solution because it will backtrack over moves that did not necessarily directly lead to an error, as well as ones that did. In this case, nonchronological backtracking, or dependency-directed backtracking could be more useful because it could identify the steps earlier in the search tree that caused the problem further down the tree.

GAME PLAYING 2

PREVIOUS YEARS QUESTIONS

PART-A

Q1. Write the features of game playing techniques.

Ans. They provide a structured task in which it is very easy to measure success or failure. They did not obviously require large amounts of knowledge.

Q2. What is Alpha - Beta Cutoff?

Ans. Alpha Beta Cutoff : This is a modified version of minimax algorithm, where in two threshold values are maintained for future expansion.

Q3. Define minimax procedure in short.

Ans. Minimax procedure is a straight forward recursive procedure that relies on two auxiliary procedures that are specific to the game being played.

Q4. What do you mean by singular extensions?

Ans. One particularly successful form secondary search is singular extensions. The idea behind singular extension is that the node is expanded one extra ply if a leaf node is found superior to its siblings and if the value of the entire search depends critically on the correctness of that node's value.

PART-B

Prob.5 Enumerate classical "Water Jug Problem". Describe the state space for this problem. Solve this problem by giving its operation sequence.

[R.T.U. 2019, 2016, 2012, R.U. 2005, 2003]

OR

Write production rules and give one solution for solving following 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 gallon of water into the 4-gallon jug?"

[R.T.U. 2010]

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

The state space for this problem can be described as the set of ordered pairs of integers (x, y) , such that $x = 0, 1, 2, 3$ or 4 and $y = 0, 1, 2$ or 3 ; x represents the number of gallons of water in the 4-gallon jug, and y represents the quantity of water in the 3-gallon jug. The start state is $(0, 0)$. The goal state is $(2, n)$ for any value of n (since the problem does not specify how many gallons need to be in the 3-gallon jug).

The operators to be used to solve the problem can be described as shown in Fig. 1. We have assumed that we can fill a jug from the pump, that we can pour water out of a jug onto the ground, that we can pour water from one jug to another and that there are no other measuring devices available. Additional assumptions such as these are almost

Artificial Intelligence

always required when converting from a typical problem statement given in English to a formal representation of the problem suitable for use by a program.

To solve the water jug problem, all we need, in addition to the problem description given above, is a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, the appropriate change to the state is made as described in the corresponding right side and the resulting state is checked to see if it corresponds to a goal state. As long as it does not, the cycle continues. Clearly the speed with which the problem gets solved depends on the mechanism that is used to select the next operation to be performed.

For the water jug problem, as with many others, there are several sequences of operators that solve the problem. One such sequence is shown in Fig.2. Often, a problem contains the explicit or implied statement that the shortest (or cheapest) such sequence be found. If present, this requirement will have a significant effect on the choice of an appropriate mechanism to guide the search for a solution.

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

Gallons in the 4-Gallon Jug	Gallons in the 3-Gallon Jug	Rule Applied
0	0	2
0	3	9
3	0	2
3	3	7
4	2	5 or 12
0	2	9 or 11
2	0	1

Fig. 2 : One Solution to the Water Jug Problem

Prob.6 Why does the search in game playing programs always proceed forward from the current position rather than backward from a goal state? [R.T.U. 2016, 2014]

Sol. In game playing programs two things be done
(1) Improve the generate procedure so that only good moves are generated.
(2) Improve the test procedure so that best moves will be recognized and explored first.

In game playing, as in other problem domains, search is not the only available technique. In some games there are at least some times when more direct techniques are appropriate. The ideal way to use a search procedure to find a solution to a problem is to generate moves through the problem space until a goal state is reached. In the context of game playing programs, a goal state is one in which we win. Unfortunately it is not usually possible, even with a good possible move generated, to search until a goal state is found. Deciding which moves have contributed to wins and which to losses is not always easy. Suppose we make a very bad move but then, because the opponent makes a mistake, we ultimately win the game. So it is not possible to decide prior the goal states and to move backward in game playing. We always have to move in forward direction.

Prob.7 Why does Game playing appear to be good domain for exploring machine intelligence? Explain its importance feature compared to ordinary AI application. [Raj. Univ. 2007]

highly stylized, so they are best played by table lookup into a database of stored patterns. To play an entire game then, we need to combine search oriented and non-search oriented techniques.

Prob.8 Explain chess problem.

Sol. Playing Chess: Defining the problem as state space search. Let's build a program that could "Play Chess", we would first have to specify the starting position of the chess board, the rules that define the legal moves, and the board positions that represent a win for one side or the other. In addition, we must make explicit the previously implicit goal of not only playing a legal game of chess but also winning a the game, if possible.

For the problem "Play Chess," it is fairly easy to provide a formal and complete problem description. The starting position can be described as an 8-by-8 array where each position contains a symbol standing for the appropriate piece in the official chess opening position. We can define as our goal any board position in which the opponent does not have a legal move and his or her king is under attack. The legal moves provide the way of getting from the initial state to a goal state. They can be described easily as a set of rules consisting of two parts: a left side that serves as a pattern to be matched against the current board position and a right side that describes the change to be made to the board position to reflect the move. There are several ways in which these rules can be written. For example, we could write a rule such as that shown in Fig.1.

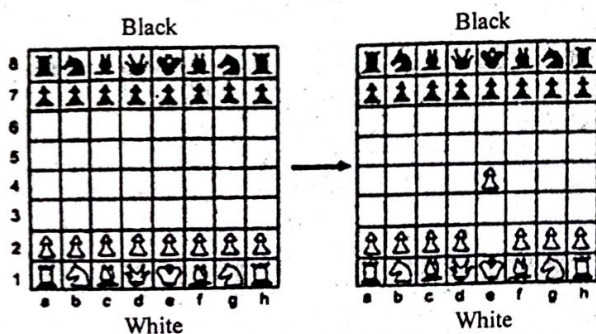


Fig. 1 : One Legal Chess Move

However, if we write rules like the one above, we have to write a very large number of them since there has to be a separate rule for each of the roughly 10^{120} possible board positions. Using so many rules poses two serious practical difficulties:

- No person could ever supply a complete set of such rules. It would take too long and could certainly not be done without mistakes.

Artificial Intelligence

- No program could easily handle all those rules. Although a hashing scheme could be used to find the relevant rules for each move fairly quickly, just storing that many rules poses serious difficulties.

In order to minimize such problems, we should look for a way to write the rules describing the legal moves in as general a way as possible. To do this, it is useful to introduce some convenient notation for describing patterns and substitutions. For example, the rule described in Fig.1, as well as many like it, could be written as shown in Fig.2. In general, the more succinctly we can describe the rules we need, the less work we will have to do to provide them and the more efficient the program that uses them can be.

White pawn at
Square (file e, rank 2)
AND
Square (file e, rank 3)
is empty
AND

→ move pawn from
Square (file e, rank 2)
to Square (file e, rank 4)

Square (file e, rank 4)
is empty

Fig. 2 Another Way to Describe Chess Moves

We have just defined the problem of playing chess as a problem of moving around in a state space, where each state corresponds to a legal position of the board. We can then play chess by starting at an initial state, using a set of rules to move from one state to another, and attempting to end up in one of a set of final states. This state space representation seems natural for chess because the set of states, which corresponds to the set of board positions, is artificial and well-organized. This same kind of representation is also useful for naturally occurring, less well-structured problems, although it may be necessary to use more complex structures than a matrix to describe an individual state. The state space representation forms the basis of most of the AI methods we discuss here. Its structure corresponds to the structure of problem solving in two important ways:

- It allows for a formal definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operations.
- It permits us to define the process of solving a particular problem as a combination of known techniques (each represented as a rule defining a single step in the space) and search, the general technique of exploring the space to try to find some path from the current state to a goal state. Search is a very important process in the solution of hard problems for which no more direct techniques are available.

Prob 9 In addition to Alpha-Beta pruning what are the modifications to the minimax procedure that can improve its performance?

OR

What is alpha - beta planning strategy? Explain its need with example.

OR

What is the need of alpha - beta strategy? Explain with example.

OR

What is "alpha-beta pruning" show by a suitable example how alpha-beta procedure works briefly comment on alpha-beta procedure effectiveness. What are the roles of following components-

"Static evaluation function generator"

"Plausible move generator"

for game playing program?

[R.T.U. 2010, Raj Univ. 2005, 2003]

Sol. 1. MOVEGEN(Position, Player) : The plausible-move generator, which returns a list of nodes representing the moves that can be made by Player in Position. We call the two players PLAYER-ONE and PLAYER-TWO. In a chess program, we might use the names BLACK and WHITE instead.

2. STATIC (Position, Player) : The static evaluation function, which returns a number representing the goodness of Position from the standpoint of Player.

It is necessary to modify the branch-and-bound strategy to include two bounds, one for each of the players. This modified strategy is called alpha-beta pruning. It requires the maintenance of two threshold values, one representing a lower bound on the value that a maximizing node may ultimately be assigned (we call this alpha), and another representing an upper bound on the value that a minimizing node may be assigned (this we call beta).

To see how the alpha-beta procedure works, consider the example shown in Fig. 1 after examining node F, we know that the opponent is guaranteed a score of 5 or less, at C (since the opponent is the minimizing player). But we also know that we are guaranteed a score of 3 or greater at node A, which we can achieve if we move to B. Any other move to B produces a score of less than 3, is worse than the move to B, and we can ignore it. After examining only F, we are sure that a move to C is worse regardless of the score of node G. Thus we need not bother to explore node G at all. Of course, cutting

AL26

out one node may not appear to justify the expense of keeping track of the limits and checking them, but if we were exploring this tree to six ply, then we would have eliminated not a single node but an entire tree three ply deep.

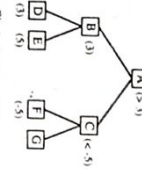


Fig. 1: An Alpha-Cut-off

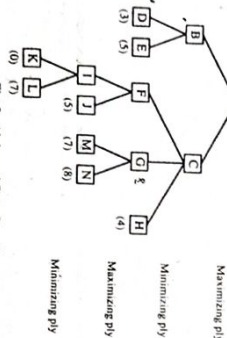


Fig. 2: Alpha and Beta Cutoffs

To see how the two thresholds, alpha and beta, can both be used, consider the example shown in Fig. 2. In searching this tree, the entire subtree headed by B is searched and we discover that at A we can expect a score of at least 3. When this alpha value is passed down to F, it will enable us to skip the exploration of L. Let's see why. After K is examined, we see that I is guaranteed a maximum score of 0, which means that F is guaranteed a minimum of 0. But this is less than alpha's value of 3, so no more branches of F need be considered. The maximizing player already knows not to choose to move to C and then to I since, if that move is made, the resulting score will be no better than 0 and a score of 3 can be achieved by moving to B instead. Now let's see how the value of beta can be used. After cutting off further exploration of I, J is examined, yielding a value of 5, which is assigned as the value of F (since it is the maximum of 5 and 0). This value becomes the value of beta at node C. It indicates that C is guaranteed to get a 5 or less. Now we must expand G. First M is examined and it has a value of 7, which is passed back to G as its tentative value. But now 7 is compared to

B.Tech. (IT Sem.) C.S. Solved Papers

beta (5). It is great and the player whose turn it is at node C is trying to minimize. So this player will not choose G, which would lead to a score of at least 7, since there is an alternative move to F, which will lead to a score of 5. Thus it is not necessary to explore any of the other branches of G.

From this example, we see that at maximizing levels, we can rule out a move early if it becomes clear that its value will be less than the current threshold, while at minimizing levels, search will be terminated if values that are greater than the current threshold are discovered. But ruling out a possible move by a maximizing player actually means cutting off the search at a minimizing level! Look again at the example in Fig. 1. Once we determine that C is a bad move from A, we cannot bother to explore G or any other paths, at the minimizing level below C. So the way alpha and beta are actually used is that search at a minimizing level can be terminated when a value less than alpha is discovered, while a search at a maximizing level can be terminated when a value greater than beta has been found. Cutting off search at a maximizing level when a high value is found may seem counter-intuitive at first, but if you keep in mind that we only get to a particular node at a maximizing level if the minimizing player at the level above chooses it, then it makes sense.

Prob. 10 Explain the algo of minimax search procedure with suitable diagram of two ply search and backing up the value of two ply search.

OR

What are game playing techniques? Explain minimax procedure with example.

[R.T.U. 2019, 2018, 2017]

OR

Explain the algo of minimax search procedure with suitable diagram of two ply search and backing up the value of two ply search.

OR

What are game playing techniques? Explain minimax procedure with example.

[R.T.U. 2019]

Sol. Game Playing Techniques :

- They provide a structured task in which it is very easy to measure success or failure.
- They did not obviously require large amounts of knowledge.
- They were thought to be solvable by straight forward search from the starting state to a winning position are called game playing techniques by machine.

Artificial Intelligence

Example (The Minimax Search Procedure) : The minimax search procedure is a depth-first, depth-limited search procedure. The idea is to start at the current position and use the plausible move generator to generate the set of possible successor positions. Now we can apply the static evaluation function to those positions and simply choose the best one. After doing so, we can back that value up to the starting position to represent our evaluation of it. The starting position is exactly as good for us as the position generated by the best move we can make next. Here we assume that the static evaluation function returns large values to indicate good situations for us, so our goal is to maximize the value of the static evaluation function at the next board position.

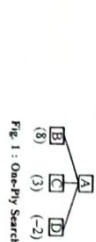


Fig. 1: One-Ply Search

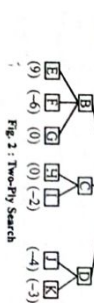


Fig. 2: Two-Ply Search

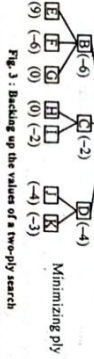


Fig. 3: Backing up the values of a two-ply search

An example of this operation is shown in Fig. 1. It assumes a static evaluation function that returns values ranging from -10 to 10, with 10 indicating a win for us, -10 a win for the opponent and 0 an even match. Since our goal is to maximize the value of the heuristic function, we choose to move to B. Backing B's value up to A, we can conclude that A's value is 8, since we know we can move to a position with a value of 8.

But since we know that the static evaluation function is not completely accurate, we would like to carry the search farther ahead than one ply. We would like to look ahead to see what will happen to each of the new game positions at the next move which will be made by the opponent. Instead of applying the static evaluation function to each of the positions that we just generated, we apply the plausible-move generator, generating a set of successor positions for each position. If we wanted to stop here, at two-ply look

AL27

ahead, we could apply the static evaluation function to each of these positions, as shown in Fig. 2.

But now we must take into account that the opponent gets to choose which successor moves to make and thus which terminal value should be backed up to the next level. Suppose we made move B. Then the opponent must choose among moves E, F and G. The opponent's goal is to minimize the value of the evaluation function, so he or she can be expected to choose move F. This means that if we make move B, the actual position in which we will end up one move later is very bad for us. This is true even though a possible configuration is that represented by node E, which is very good for us. But since at this level we are not the ones of propagating the new values up the tree, at the level representing the opponent's choice, the minimum value was chosen and backed up. At the level representing our choice, the maximum value was chosen.

Minimax procedure is a straightforward recursive procedure that relies on two auxiliary procedures that are specific to the game being played:

1. MOVEGEN (Position, Player) : The plausible move generator, which returns a list of nodes representing the moves that can be made by player in Position. We call the two players PLAYER-ONE and PLAYER-TWO. In a chess program, we might use the names BLACK and WHITE instead.
2. STATIC (Position, Player) : The static evaluation function, which returns a number representing the goodness of Position from the standpoint of Player.

Algorithm : MINIMAX (Position, Depth, Player) :

1. IF DEEP-ENOUGH (Position, Depth), then return the structure
- VALUE = STATIC (Position, Player);
- PATH = null

This indicates that there is no path from this node and that its value is that determined by the static evaluation function.

2. Otherwise, generate one more ply of the three by calling the function MOVE-GEN (Position Player) and setting SUCCESSORS to the list it returns.
3. IF SUCCESSORS is empty, then there are no moves to be made, so return the same structure that would have been returned IF DEEP-ENOUGH had returned true.

4. If SUCCESSORS is not empty, then examine each element in turn and keep track of the best one. This is done as follows :

Initialize BEST-SCORE to the minimum value that STATIC can return. It will be updated to reflect the best score that can be achieved by an element of SUCCESSORS.

For each element SUCCESSORS, do the following :

- (a) Set RESULT-SUCC to

MINIMAX (SUCC, Depth + 1, OPPOSITE (Player))

This recursive call to MINIMAX will actually carry out the exploration of SUCC.

- (b) Set NEW-VALUE to VALUE (RESULT-SUCC). This will cause it to reflect the merits of the position from the opposite perspective from that of the next lower level.

- (c) If NEW-VALUE < BEST-SCORE, then we have found a successor that is better than any that have been examined so far. Record this by doing the following :

- (i) Set BEST-SCORE to NEW-VALUE.

- (ii) The best known path is now from CURRENT to SUCC and then on to the appropriate path down from SUCC as determined by the recursive call to MINIMAX. So set BEST-PATH to the result of attaching SUCC to the front of PATH(RESULT.SUCC).

5. Now that all the successors have been examined, we know the value of position as well as which path to take from it. So return the structure

VALUE = BEST-SCORE

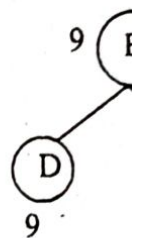
PATH = BEST-PATH

when the initial call to MINIMAX returns, the best move from CURRENT is the first element on PATH. Its execution can be traced from the game tree shown previously.

(a) Alpha:
mini
are n

One repr
maximizing r
representing a
node may be a

To explai
structures in fi



Here the
minimizer. As

The max
value at D ar
maximizer is
moves to B.

Now, co
-5 and at G is
by moving to
totally unacce
to have a val
pruning the t
searching.

(b) Seconda
horizon effec
sure that a hi
away from th
game tree up
particular mc
have been too
of eight, it is
branch an ad

AI.29

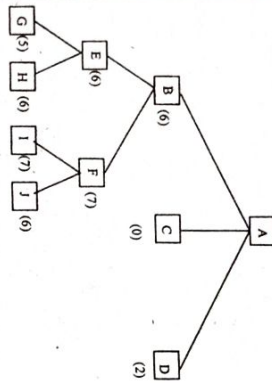


Fig. 4 : The Calm Situation

The horizon effect may also influence a program's perception of good moves. It can make a move look good despite the fact that the move might be better if delayed past the horizon. Even with quiescence, all fixed-depth search programs are subject to subtle horizon effects.

Prob.12 Explain 8 tile puzzle problem specification in detail.

Sol. Eight-puzzle Problem: A block of 8 tiles is given in a certain position. One tile could be moved at the vacant position and the desired goal state is to be achieved.

2	8	3	1	2	3
1	6	4	8	-	4
7	-	5	7	6	5

Initial State

1	2	3
8	-	4
7	6	5

Goal State

The state space representation for 8-puzzle problem.
The heuristic used for the solution is:

See if tile is in its right position i.e. at the position of the Goal State. If so then assign weight (tile) = 1. If not then don't assign any weight to it. Now add the weight of all tiles and select the next state as the situation with maximum weight. This is because higher the weight more number of tiles are in their correct positions and we can get the Goal state.

AI.30

Now here there can be maximum 4 possible moves i.e. move1, move2, move3, move4.
So the production rules are:

1	(move1, move2, move3, move4) → (move1) if wt(move1) > wt(move2) & wt(move3) & wt(move4).	if wt on move1 is greater than wt on other three moves then choose move1
2	(move1, move2, move3, move4) → (move2) if wt(move2) > wt(move1) & wt(move3) & wt(move4).	if wt on move2 is greater than wt on other three moves then choose move2
3	(move1, move2, move3, move4) → (move3) if wt(move3) > wt(move1) & wt(move2) & wt(move4).	if wt on move3 is greater than wt on other three moves then choose move3
4	(move1, move2, move3, move4) → (move4) if wt(move4) > wt(move1) & wt(move2) & wt(move3).	if wt on move4 is greater than wt on other three moves then choose move4
5	(move1, move2, move3, move4) → (move1 or move2) if wt(move1) = wt(move2) & wt(move3) & wt(move4) < wt(move1) & wt(move2).	if wt on any two moves are equal then choose the move with more wt on it and if it leads to a dead end then back track and choose other
6	(move1, move2, move3, move4) → (move3 or move4) if wt(move3) = wt(move4) & wt(move1) & wt(move2) < wt(move3) & wt(move4).	if wt on any two moves are equal then choose the move with more wt on it and if it leads to a dead end then back track and choose other

B.Tech. (VI Sem.) C.S. Solved Papers

7	(move1, move2, move3, move4) → (move1 or move3) if wt(move1) = wt(move3) & wt(move2) & wt(move4) > wt(move1) & wt(move3).	if wt on any two moves are equal then choose the move with more wt on it if it leads to a dead end then back track and choose other
8	(move1, move2, move3, move4) → (move2 or move4) if wt(move2) = wt(move4) & wt(move1) & wt(move3) > wt(move2) & wt(move4).	if wt on any two moves are equal then choose the move with more wt on it. If it leads to a dead end then back track and choose other.
9	(move1, move2, move3, move4) → (move1 or move4) if wt(move1) = wt(move4) & wt(move2) & wt(move3) < wt(move1) & wt(move4).	if wt on any two moves are equal then choose the move with more wt on it. If it leads to a dead end then back track and choose other
10	(move1, move2, move3, move4) → (move2 or move3) if wt(move2) = wt(move3) & wt(move1) & wt(move4) < wt(move2) & wt(move3).	if wt on any two moves are equal then choose the move with more wt on it. If it leads to a dead end then back track and choose other

Now lets apply these rules to solve below problem:

Initial State:

1	8	3
7	6	4
7	-	5

Goal State

1	2	3
8	-	4
7	6	5

Artificial Intelligence

AI.31

Step 4:

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

Step 1: Move 1

Move 2

Move 3

Move 4

Move 5

Move 6

Move 7

Move 8

Move 9

Move 10

Move 11

Move 12

Move 13

Move 14

Move 15

Move 16

Move 17

Move 18

Move 19

Move 20

Move 21

Move 22

Move 23

Move 24

Move 25

Move 26

Move 27

Move 28

Move 29

Move 30

Move 31

Move 32

Move 33

Move 34

Move 35

Move 36

Move 37

Move 38

Move 39

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

Step 5: Move 1

Move 2

Move 3

Move 4

Move 5

Move 6

Move 7

Move 8

Move 9

Move 10

Move 11

Move 12

Move 13

Move 14

Move 15

Move 16

Move 17

Move 18

Move 19

Move 20

Move 21

Move 22

Move 23

Move 24

Move 25

Move 26

Move 27

Move 28

Move 29

Move 30

Move 31

Move 32

Move 33

Move 34

Move 35

Move 36

Move 37

Move 38

Move 39

Sol. Probability provides the way of summarizing the uncertainty that comes from our laziness and ignorance. Probability statements do not have quite the same kind of semantics known as evidences.

Prob.5 What is the need for utility theory in uncertainty?

Sol. Utility theory says that every state has a degree of usefulness, or utility to In agent, and that the agent will prefer states with higher utility. The use utility theory to represent and reason with preferences.

PART-B

Prob.6 Explain the difference between backward and forward reasoning. [R.T.U. 2016]

OR

Differentiate forward and backward reasoning.

[R.T.U. 2019, 2013]

OR

Explain the difference between forward and backward reasoning (Chaining) & under what condition each would be best to use for. [R.T.U. 2012, Raj. Univ. 2005]

OR

Write the short note on forward and backward Reasoning. [R.T.U. 2010, Raj. Univ. 2007]

Sol. The object of a search procedure is to discover a path through a problem space from an initial configuration to a goal state.

Artificial Intelligence

There are actually two directions in which such a search could proceed :

- Forward, from the start states.
- Backward, from the goal states.

The production system model of search process provides an easy way of viewing forward and backward reasoning as symmetric processes. Consider the problem of solving a particular instance of the 8-Puzzle. Two rules should be used.

(A) Reason Forward from the Initial States : Begin building a tree of move sequences that might be solutions by starting with the initial configuration(s) at the root of the tree. Generate the next level of the tree by finding all the rules whose left sides watch the root node and using their right sides to create the new configurations. Generate the next level by taking a node generated at the previous level and applying to it all of the rules whose left sides watch it continue until a configuration that matches the goal state is generated.

(B) Reason Backward from the Goal States : Begin building a tree of move sequences that might be solutions by starting with goal configuration(s) at the root of the tree. Generate the next level of the tree by finding all the rules whose right sides match the root node. These are all the rules that, if only we could apply them, would generate the state we want. Use the left sides of the rules to generate the nodes at this second level of the tree. Generate the next level of the tree by taking each node at the previous level and finding all the rules whose right sides match it. Then use the corresponding left sides to generate the new nodes. Continue until a node that matches in initial state is generated. This method of reasoning backward from in desired final state is often called goal directed reasoning.

Factors effecting the forward and backward reasoning and choosing the better one are :

- Are there more possible start states or goal states ? We would like to move from the smaller set of states to the larger set of states.
- In which direction is the branching factor greater where branching factor is the average member of nodes that can be reached directly from a single node ? We would like to proceed in the direction with the lower branching factor.
- Will the program be asked to justify its reasoning process to a user ? If so, it is important to proceed in the direction that corresponds more closely with the way the user will think.
- What kind of event is going to trigger a problem. Solving episode ? If it is the arrival of a new fact, forward reasoning makes sense. If it is a query to which a response is desired, backward reasoning is more natural.

Prob.7 Define the following terms :

- (i) Mapping (ii) Homomorphic
(iii) Horn clause (iv) Reasoning [R.T.U. 2019, 2012]

Sol. (i) **Mapping** : Maps ultimately help your AI make better decisions by providing useful information about the world. In particular, influence maps provide three different types of information that are particularly useful for decision making :

- (a) **Situation Summary** : Influence maps do a great job of summarizing all the little details in the world and making them easy to understand at a glance. Who's in control of what area? Where are the borders between the territories? How much enemy presence is there in each area?

(b) **Historical Statistics** : Beyond just storing information about the current situation, maps can also remember what happened for a certain period of time. Was this area being assaulted? How well did my previous attack go?

(c) **Future Predictions** : An often ignored aspect of maps, they can also help predict the future. Using the map of the terrain, you can figure out where an output would go and how its influence would extend in the future.

Sol. (ii) **Homomorphic** : By using this method, multiple servers/evaluators cooperatively perform dynamic programming procedures for solving a combinatorial optimization problem by using the private information sent from agents as inputs. Although the evaluators can compute the optimal solution correctly, the inputs are kept secret even from the servers. Furthermore, we discussed its application to several combinatorial auctions, i.e., multi-unit auctions, linear-good auctions and general combinatorial auctions.

Sol. (iii) **Horn clause** : A Horn clause is either a definite clause or an integrity constraint. That is, a Horn clause has either false or a normal atom as its head. Integrity constraints allow the system to prove that some conjunction of atoms is false in all models of a knowledge base - that is, to prove disjunctions of negations of atoms. Recall that $\neg p$ is the negation of p , which is true in an interpretation when p is false in that interpretation and $p \vee q$ is the disjunction of p and q , which is true in an interpretation if p is true or q is true or both are true in the interpretation. The integrity constraint false $\leftarrow \text{all } \vee \dots \vee \text{all}$ is logically equivalent to $\neg \text{all } \vee \dots \vee \text{all}$.

Sol. (iv) **Reasoning** : To reason is to draw inferences appropriate to the situation. Inferences are classified as either deductive or inductive. An example of the former is, "Fred

must be in either the museum or the cafe. He is not in the cafe, therefore he is in the museum," and of the latter, "Previous accidents of this sort were caused by instrument failure, therefore this accident was caused by instrument failure." The most significant difference between these forms of reasoning is that in the deductive case the truth of the premises guarantees the truth of the conclusion, whereas in the inductive case the truth of the premise lends support to the conclusion without giving absolute assurance. Inductive reasoning is common in science, where data are collected and tentative models are developed to describe and predict future behavior, until the appearance of anomalous data forces the model to be revised. Deductive reasoning is common in mathematics and logic, where elaborate structures of irrefutable theorems are built up from a small set of basic axioms and rules.

Prob.8 Explain Dempster-Shafer Theory.

[R.T.U. 2018]

Sol. **Dempster-Shafer theory** : The theory of belief functions, also referred to as evidence theory or Dempster-Shafer theory (DST), is a general framework for reasoning with uncertainty, with understood connections to other frameworks such as probability, possibility and imprecise probability theories. First introduced by Arthur P. Dempster in the context of statistical inference, the theory was later developed by Glenn Shafer into a general framework for modeling epistemic uncertainty, a mathematical theory of evidence. The theory allows one to combine evidence from different sources and arrive at a degree of belief (represented by a mathematical object called belief function) that takes into account all the available evidence.

In a narrow sense, the term Dempster-Shafer theory refers to the original conception of the theory by Dempster and Shafer. However, it is more common to use the term in the wider sense of the same general approach, as adapted to specific kinds of situations. In particular, many authors have proposed different rules for combining evidence, often with a view to handling conflicts in evidence better. The early contributions have also been the starting points of many important developments, including the transferable belief model and the theory of hints.

Dempster-Shafer theory is a generalization of the Bayesian theory of subjective probability. Belief functions

base degrees of belief (or confidence, or trust) for one question on the probabilities for a related question. The degrees of belief themselves may or may not have the mathematical properties of probabilities; how much they differ depends on how closely the two questions are related. Put another way, it is a way of representing epistemic plausibilities but it can yield answers that contradict those arrived at using probability theory.

Often used as a method of sensor fusion, Dempster-Shafer theory is based on two ideas: 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. In essence, the degree of belief in a proposition depends primarily upon the number of answers (to the related questions) containing the proposition, and the subjective probability of each answer. Also contributing the rules of combination that reflect general assumptions about the data.

In this formalism a degree of belief (also referred to as a mass) is represented as a belief function rather than a Bayesian probability distribution. Probability values are assigned to sets of possibilities rather than single events; their appeal rests on the fact they naturally encode evidence in favor of propositions.

Dempster-Shafer theory assigns its masses to all of the subsets of the propositions that compose a system—in set-theoretic terms, the power set of the propositions. For instance, assume a situation where there are two related questions, or propositions, in a system. In this system, any belief function assigns mass to the first proposition, the second, both or neither.

Prob.9 What is predicate logic? Differentiate propositional and predicate logic.

[R.T.U. 2017, 2016, 2013]

OR

Explain the difference between propositional and predicate logic? [R.T.U. 2015]

Sol. Predicate logic is a useful way of representing and manipulating some of the kinds of knowledge using real world facts as statements written as WFF's (Well-Formed Formulas).

S. No.	Propositional logic	Predicate logic
1	A way of representing the sort of world knowledge.	A way of representing real world facts as statements written as WFF's (well formed formula) difficult.
2	It is simple to deal.	If permits representations of things that cannot reasonably be represented in propositional logic.
3	Less efficient	Use variable and quantities.
4	Use statements or proposition.	Use variable and quantities.
5	Example • It is raining RAINING • It is sunny SUNNY • If it is raining, then it is not sunny RAINING \rightarrow SUNNY • Dheer is a man DHEERMAN • All man are mortal MORTALMAN	Example • Dheer is a man man(Dheer) • Ram was a king king (Ram) • Everyone is loyal to someone $Vx : \exists y : \text{loyal to } (x, y)$ • Marcus tried to assassinate caesar (trassinate, caesar) • All man are mortal $Vx : \text{man}(x) \rightarrow \text{mortal}(x)$
6	They represent true or false logic.	The represent mathematical terms.
7	True/False values for it do not depend on any parameter like $2+3=5$.	While predicate is statement whose truth value depends on the parameters specified in the statement like $X-3=5$ is a parameter here.
8	Here propositions are symbolized.	Here both proposition and predicates are symbolized.
9	It uses composite formulas like (Not A.B).	While here quantifiers like "There Exist" or "For Every" are used.
10	Propositional logic is decidable.	The full predicate logic is undecidable.

[R.T.U. 2017, 2012]

Sol. **Minimalist Reasoning** : We describe methods for saying a very specific and highly useful class of things that are generally true. These methods are based on some variant of the idea of a minimal model. Although there are several distinct definitions of what constitutes a minimal model. We will define a model to be minimal if there are no other model in which fewer things are true. The idea behind using minimal models as a basis for statement than false ones. If something is true and relevant it makes sense to assume that it has been entered into our knowledge base. Therefore, assume that the only true statements are those that necessarily must be true in order to maintain the consistency of the knowledge.

Prob.11 Compute monotonic and non monotonic reasoning.

[R.T.U. 2016]

Compute monotonic and non monotonic reasoning.

OR

[R.T.U. 2013]

Sol. **Comparison Monotonic and Non-monotonic Reasoning** : Let us understand the difference between monotonic and nonmonotonic reasoning with the help of an example.

- A regatta is a race between boats
- Boats travel on water

From these two statements it should follow that a regatta takes place on water. But not so in Alice Springs, Central Australia. The annual Todd River Regatta does not take place on water for the simple reason that the river is dry. (The boats have holes in the bottom for the crew to put their feet through.) In this special context, regattas do not take place on water.

This example explains the difference between monotonic and nonmonotonic reasoning. In monotonic reasoning, when a statement P follows from a set of statements A and additional statements are added to A , then P remains true. This is not the case with nonmonotonic reasoning. Adding new statements to A can cause P to become false. This is exactly what happens when we move from a wider context (all regattas) to a narrower context (the Todd River Regatta). Because the context now contains an additional assumption (dry riverbed), our theory about regattas comes to a different conclusion. (Prospective regatta participants should be aware that in years when the Todd River carries water the regatta is canceled.)

Prob.12 Differentiate between domain dependent knowledge and domain independent knowledge.

[R.T.U. 2012]

All men are mortal.

We could represent this as :

MORTALMAN

But that fails to capture the relationship between any individual being a man and that individual being a mortal. To do that, we really need variables and quantification unless we are willing to write separate statements about the mortality of every known man. So we appear to be forced to move to first-order predicate logic (or just predicate logic as a way of representing knowledge because it permits representations of things that cannot reasonably be represented in propositional logic. In predicate logic, we can represent real-world facts as statements written as wff's.

Prob.14 What is first order logic? Explain basic elements of first order logic.

Sol. First-Order logic

1. First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
2. FOL is sufficiently expressive to represent the natural language statements in a concise way.
3. First-order logic is also known as Predicate logic or First-order predicate logic. First order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
4. First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
 - (i) **Objects** : A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, ...
 - (ii) **Relations** : It can be unary relation such as: red, round, is adjacent, or n-ary relation such as: the sister of, brother of, has color, comes between
 - (iii) **Function** : Father of, best friend, third inning of, end of,
5. As a natural language, first-order logic also has two main parts:
 - (a) Syntax
 - (b) Semantics

Syntax of First-Order logic : The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in shorthand notation in FOL.

Basic Elements of First-order logic: Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,
Variables	x, y, z, a, b,
Predicates	Brother, Father, >,
Function	sqrt, LeftLegOf,
Connectives	$\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$
Equality	$=$
Quantifier	\forall, \exists

Prob.15 Define partial order planning.

Sol. Partial-Order Planning : The forward and regression planners enforce a total ordering on actions at all stages of the planning process. The CSP planner commits to the particular time that the action will be carried out. This means that those planners have to commit to an ordering of actions that cannot occur concurrently when adding them to a partial plan, even if there is no particular reason to put one action before another.

The idea of a partial-order planner is to have a partial ordering between actions and only commit to an ordering between actions when forced. This is sometimes also called a non-linear planner, which is a misnomer because such planners often produce a linear plan.

A partial ordering is a less-than relation that is transitive and asymmetric. A partial-order plan is a set of actions together with a partial ordering, representing a 'before' relation on actions. such that any total ordering of the actions, consistent with the partial ordering, will solve the goal from the initial state. Write $act_0 < act_1$ if action act_0 is before action act_1 in the partial order. This means that action act_0 must occur before action act_1 .

For uniformity, treat start as an action that achieves the relations that are true in the initial state, and treat finish as an action whose precondition is the goal to be solved. The pseudoaction starts is before every other action, and finish is after every other action. The use of these as actions means that the algorithm does not require special cases for the initial situation and for the goals. When the preconditions of finish hold, the goal is solved.

An action, other than start or finish, will be in a partial-order plan to achieve a precondition of an action in the plan. Each precondition of an action in the plan is either true in the initial state, and so achieved by start, or there will be an action in the plan that achieves it.

We must ensure that the actions achieve the conditions they were assigned to achieve. Each precondition P of an action act_1 in a plan will have an action act_0 associated with it such that act_0 achieves precondition P for act_1 . The triple $\{act_0, P, act_1\}$ is a causal link. The partial order specifies

that action act_0 occurs before action act_1 , which is written as $act_0 < act_1$. Any other action A that makes P false must either be before act_0 or after act_1 .

Informally, a partial-order planner works as follows: Begin with the actions start and finish and the partial order start < finish. The planner maintains an agenda that is a set of (P, A) pairs, where A is an action in the plan and P is an atom that is a precondition of A that must be achieved. Initially the agenda contains pairs $(G, finish)$, where G is an atom that must be true in the goal state.

At each stage in the planning process, a pair (G, act_1) is selected from the agenda, where P is a precondition for action act_1 . Then an action, act_0 is chosen to achieve P . That action is either already in the plan - it could be the start action, for example - or it is a new action that is added to the plan. Action act_0 must happen before act_1 in the partial order. It adds a causal link that records that act_0 achieves P for action act_1 . Any action in the plan that deletes P must happen either before act_0 or after act_1 . If act_0 is a new action, its preconditions are added to the agenda, and the process continues until the agenda is empty.

This is a non-deterministic procedure. The "choose" and the "either ... or ..." form choices that must be searched over. There are two choices that require search:

- which action is selected to achieve G and
- whether an action that deletes G happens before act_0 or after act_1 .

PART-C

Prob.16 What is knowledge representation and also differentiate knowledge and knowledge base?
[R.T.U. 2019]

OR

What is knowledge representation? Explain its approaches and issues.
[R.T.U. 2016]

OR

What are the various approaches and issues in knowledge representation?
[R.T.U. 2019]

OR

Discuss various approaches and issues in knowledge representation?
[R.T.U. 2015, 2012, Raj Univ. 2004]

OR

Explain issues in knowledge representation.
[R.T.U. 2014]

OR

What is knowledge representation? What are the problems facing representation knowledge?
[R.T.U. 2011]

Sol. A good system for the representation of knowledge in a particular domain should possess the following four properties:

1. **Representational Adequacy**: The ability to represent all of the kinds of knowledge that are needed in that domain.

2. **Inferential Adequacy**: The ability to manipulate the representational structures in such a way as to derive new structures corresponding to new knowledge inferred from old.

3. **Inferential Efficiency**: The ability to incorporate into the knowledge structure additional information that can be used to focus the attention of the inference mechanisms in the most promising directions.

4. **Acquisitional Efficiency**: The ability to acquire new information easily. The simplest case involves direct insertion, by a person, of new knowledge into the database. Ideally, the program itself would be able to control knowledge acquisition. Unfortunately, no single system that optimizes all of the capabilities for all kinds of knowledge has yet been found. As a result, multiple techniques for knowledge representation exist. Many programs rely on more than one technique.

Simple Relational Knowledge

The simplest way to represent declarative facts is as a set of relations of the same sort used in database systems.

Fig. 1 shows an example of such a relational system.

Player	Height	Weight	Bats-Throws
Hank Aaron	6-0	180	Right-Right
Willie Mays	5-10	170	Right-Right
Babe Ruth	6-2	215	Left-Left
Ted Williams	6-3	205	Left-Right

Fig. 1 : Simple Relational Knowledge

The reason that this representation is simple is that standing alone it provides very weak inferential capabilities. But knowledge represented in this form may serve as the input to more powerful inference engines. For example, given just the facts of figure, it is not possible even to answer the simple question, "Who is the heaviest player?" But if a procedure for finding the heaviest player is provided, then these facts will enable the procedure to compute an answer. If, instead, we are provided with a set of rules for deciding which hitter to put up against a given pitcher (based on right- and left-handedness, say), then this same relation can provide at least some of the information required by those rules.

Providing support for relational knowledge is what database systems are designed to do. Thus we do not need to discuss this kind of knowledge representation structure further here. The practical issues that arise in linking a

database system that provides this kind of support to a knowledge representation system that provides some of the other capabilities that we are about to discuss have already been solved in several commercial products.

Inheritable Knowledge

The relational knowledge corresponds to a set of attributes and associated values that together describe the objects of the knowledge base. Knowledge about objects, their attributes and their values need not be as simple as that shown in our example. In particular, it is possible to augment the basic representation with inference mechanisms that operate on the structure of the representation. For this to be effective, the structure must be designed to correspond to effective, mechanisms that are desired.

In order to support property inheritance, objects must be organized into classes and classes must be arranged in a generalization hierarchy. Some additional baseball knowledge inserted into a structure that is so arranged. Lines represent attributes. Boxed nodes represent objects and values of attributes of objects. These values can also be viewed as attributes of objects. The values shown in the fig. 3 corresponding attribute line. The structure shown in the fig. 3 is a slot-and-filter structure. It may also be called a semantic network or a collection of frames. In the latter case, each individual frame represents the collection of attributes and values associated with a particular node.

There is so much flexibility in the way that this can be used to solve particular representation problems that it is difficult to reserve precise words for particular representations. Usually the use of the term frame system implies somewhat more structure on the attributes and the inference mechanisms that are available to apply to them than does the term semantic network.

All of the objects and most of the attributes shown in this example have been chosen to correspond to the baseball domain and they have no general significance. The two exceptions to this are the attribute isa, which is being used to show class inclusion, and the attribute instance, which is being used to show class membership. These two specific (and generally useful) attributes provide the basis for property inheritance as an inference technique. Using this technique, the knowledge base can support retrieval both of facts that have been explicitly stored and of facts that can be derived from those that are explicitly stored.

Inferential Knowledge

Property inheritance is a powerful form of inference, but it is not the only useful form. Sometimes all the power of traditional logic and sometimes even more than that.

Of course, this knowledge is useless unless there is also an inference procedure that can exploit it (just as the default knowledge in the previous example would have been useless

without our algorithm for moving through the knowledge structure). The required inference procedure now is one that implements the standard logical rules of inference. There are many such procedures, some of which reason forward from given facts to conclusions, others of which reason backward from desired conclusions to given facts. One of the most commonly used of these procedures is resolution, which exploits a proof by contradiction strategy.

Logic provides a powerful structure in which to describe relationships among values. It is often useful to combine this, or some other powerful description language, with an isa hierarchy. The techniques we are describing here should not be regarded as complete and incompatible ways of representing knowledge. Instead, they should be viewed as building blocks of a complete representational system.

Procedural Knowledge

So far, our examples of baseball knowledge have concentrated on relatively static, declarative facts. But another, equally useful, kind of knowledge is operational, or procedural knowledge, that specifies what to do when. Procedural knowledge can be

Baseball-Player

```
isa: Adult-Male
bats: (lambda (x))
(prog ()
  (L1
    (cond ((caddr x) (return (caddr x)))
          (t (seq x (eval (caddr x)))
              (cond (x (go L1))
                    (t (return nil)))
              height
              batting average
              252
              6-1
```

Fig. 2 : Using LISP Code to Define a Value

Issues in Knowledge Representation

Issues that cut across all of them:

- Are any attributes of objects so basic that they occur in almost every problem domain? If there are, we need to make sure that they are handled appropriately in each of the mechanisms we propose. If such attributes exist, what are they?
 - Are there any important relationships that exist among attributes of objects?
 - At what level should knowledge be represented? Is there a good set of primitives into which all knowledge can be broken down? Is it helpful to use such primitives?
 - How should sets of objects be represented?
 - Given a large amount of knowledge stored in a database, how can relevant parts be accessed when they are needed?
- We will talk about each of these questions briefly in the next five sections.

Important Attributes

There are two attributes that are of very general significance and we have already seen their use: instance and isa. These attributes are important because they support property inheritance. They are called a variety of things in AI systems, but the names do not matter. What does matter is that they represent class membership and class inclusion and that class inclusion is transitive. In slot-and-filler systems, these attributes are usually represented explicitly in a way much like that shown in Fig. 3 and 4. In logic-based systems, these relationships may be represented this way or they may be represented implicitly by a set of predicates describing particular classes.

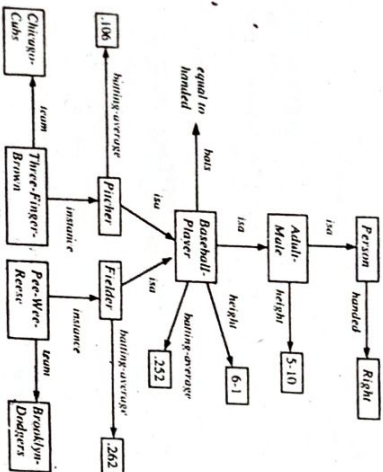


Fig. 3 : Inheritable Knowledge

Baseball-Player

isa: Adult-Male
(EQUAL, handed)
bats: 6-1
height: 252
batting average: 252

Fig. 4 : Viewing a Node as a Frame

Relationships among Attributes

The attributes that we use to describe objects are themselves entities that we represent. What properties do they have independent of the specific knowledge they encode? There are four such properties that deserve mention here:

- Inverses
- Existence in an isa hierarchy
- Techniques for reasoning about values
- Single-valued attributes

Inverses

Entities in the world are related to each other in many different ways. But as soon as we decide to describe those relationships as attributes, attributes are those relationships. So, for example, in Fig. 1, we used the attributes instance, isa and team. Each of these was shown in the Fig. 3 with a directed arrow, originating at the object that was being described and terminating at the object representing the value of the specified attribute. But we could equally well have focused on the object representing the value. If we do that, then there is still a relationship between the two entities, although it is a different one since the original relationship was not symmetric. In many cases, it is important to represent this other view of relationships. There are two good ways to do this.

The first is to represent both relationships in a single representation that ignores focus. Logical representations are usually interpreted as doing this. For example, the assertion:

team = (Pee-Wee-Reese, Brooklyn-Dodgers)

can equally easily be interpreted as a statement about Pee Wee Reese or about the Brooklyn Dodgers. How it is actually used depends on the other assertions that a system contains.

The second approach is to use attributes that focus on a single entity but to use them in pairs, one the inverse of the other. In this approach, we would represent the team information with two attributes:

- One associated with Pee Wee Reese:
team = Brooklyn-Dodgers
- One associated with Brooklyn Dodgers
team-members = Pee-Wee-Reese,...

This is the approach that is taken in semantic net and frame-based systems. When it is used, it is usually accompanied by a knowledge acquisition tool that guarantees the consistency of inverse slots by forcing them to be declared and then checking each time a value is added to one attribute that the corresponding value is added to the inverse.

An Isa Hierarchy of Attributes

Just as there are classes of objects and attributes and subsets of those classes, there are attributes and specializations of attributes. Consider, for example, the attribute height. It is actually a specialization of the more general attribute physical-size which is, in turn, a specialization of physical-attribute. These generalization-specialization relationships are important for attributes for the same reason that they are important for other concepts — they support inheritance. In the case of attributes, they support inheriting information about such things as constraints on the values that the attribute can have and mechanisms for computing those values.

Artificial Intelligence

Techniques for Reasoning about Values

Sometimes values of attributes are specified explicitly when a knowledge base is created. We saw several examples of that in the baseball example of Fig. 2. But often the reasoning system must reason about values it has not been given explicitly. Several kinds of information can play a role in this reasoning, including:

- Information about the type of the value. For example, the value of height must be a number measured in a unit of length.
- Constraints on the value, often stated in terms of related entities. For example, the age of a person cannot be greater than the age of either of that person's parents.
- Rules that describe actions that should be taken if a value ever becomes known. These rules are called forward rules, or sometimes if-added rules.

Single-Valued Attributes

A specific but very useful kind of attribute is one that is guaranteed to take a unique value. For example, a baseball player can, at any one time, have only a single height and be a member of only one team. If there is already a value present for one of these attributes and a different value is asserted, then one of two things has happened. Either a change has occurred in the world or there is now a contradiction in the knowledge base that needs to be resolved. Knowledge-representation systems have taken several different approaches to providing support for single-valued attributes, including:

- Introduce an explicit notation for temporal intervals. If two different values are ever asserted for the same temporal interval, signal a contradiction automatically.
- Assume that the only temporal interval that is of interest is now. So if a new value is asserted, replace the old value.
- Provide no explicit support. Logic-based systems are in this category. But in these systems, knowledge-base builders can add axioms that state that if an attribute has one value then it is known not to have all other values.

Difference between Knowledge and Knowledge base

Knowledge : Knowledge can be defined as the body of facts and principles accumulated by humankind or the act, fact or state of knowing. For example, in biological organisms, knowledge is stored as complex structures of interconnected neurons. The structures correspond to symbolic representations of the knowledge processed by the organisms, the fact, rules and so on. Please note that an average human brain weighs about 3.3 pounds and contains and estimated

number of 10^{12} neurons. Also note that these neurons and their interconnection capabilities provide about 10^{14} bits of potential storage capacity. On the other hand, in computers knowledge is stored as symbolic structure but in form of collections of magnetic spots and voltage states.

Knowledge consists of procedural and declarative components. Declarative knowledge is a descriptive representation of knowledge consisting of factual statements and information. These are rules and facts. Declarative knowledge is easy to acquire and document in the knowledge base.

Procedural knowledge results from the intellectual skills to do something. These skills used to make decisions are comparatively difficult to work with such knowledge. Procedural knowledge generally encompasses a sequence of actions, along with the expected results. Commonsense knowledge and heuristic knowledge are examples of procedural knowledge. The following section :

Knowledge Base : The knowledge base is the key component of a knowledge-based system. The quality and usefulness of the system is directly related to the knowledge represented in it. The knowledge base contains all types of knowledge that is given form. It is obvious that the knowledge base must contain the domain knowledge within the system intended to solve the problem. Meta-knowledge should also be stored. As stated earlier, meta-knowledge is defined as knowledge about knowledge. This generally includes knowledge about ontology—the structure in which the domain knowledge is represented, along with the target application and methods of use. Figure shows the components of the knowledge base.

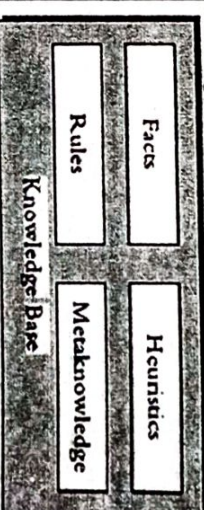


Fig. 5 : Components of knowledge base

Prob.17 What are KBS independent technologies? Explain in brief. Also write the business benefits of KBS.

[K.T.U. 2019, 2021]

Artificial Intelligence

would have floundered among the millions of possible structures.

(3) **MYCIN**: MYCIN's task was diagnosis and therapy selection for a variety of infectious diseases. It was the first system to have all of the hallmarks that came to be associated with knowledge-based systems and as such came to be regarded as a prototypical example. Its knowledge was expressed as a set of some 450 relatively independent if/then rules; its inference engine used a simple backward-chaining control structure; and it was capable of explaining its recommendations (by showing the user a recap of the rules that had been used).

Benefits

An initiative in health care would be popular and, cost effective in the long term. So the benefits if such a program was successful would be premium, there would be improved health of the population, with emphasis on prevention rather than cure, thus reductions on days lost due to illness each year.

Prob.18 Elaborate Forward and Backward chaining with algorithms. [R.T.U. 2018]

Sol. Forward Chaining: Forward chaining (or forward reasoning) is one of the two main methods of reasoning when using an inference engine and can be described logically as repeated application of *modus ponens*. Forward chaining is a popular implementation strategy for expert systems, business and production rule systems. The opposite of forward chaining is backward chaining.

Forward chaining starts with the available data and uses inference rules to extract more data (from an end user, for example) until a goal is reached. An inference engine using forward chaining searches the inference rules until it finds one where the antecedent (If clause) is known to be true. When such a rule is found, the engine can conclude, or infer, the consequent (Then clause), resulting in the addition of new information to its data.

Inference engines will iterate through this process until a goal is reached.

Forward Chaining Algorithm

function PL-FC-ENTAILS? (KB, q) returns true or false

local variables: count, a table, indexed by clause, initially the number of premises inferred, a table, indexed by symbol, each entry initial false agenda, a list of symbols, initially the symbols known to be true

while agenda is not empty do

$p \leftarrow \text{POP}(\text{agenda})$

 unless inferred [p] do

 for each Horn clause c in whose premise p appears do

 decrement count [c]

 if count[c] = 0 then do

 if HEAD[c] = q then return true

 PUSH[HEAD][c], agenda)

return false

Suppose that the goal is to conclude the color of a pet named Fritz, given that he croaks and eats flies, and that the rule base contains the following four rules:

1. If X croaks and X eats flies - Then X is a frog
2. If X chirps and X sings - Then X is a canary
3. If X is a frog - Then X is green
4. If X is a canary - Then X is yellow

Let us illustrate forward chaining by following the pattern of a computer as it evaluates the rules. Assume the following facts:

- Fritz croaks
- Fritz eats flies

With forward reasoning, the inference engine can derive that Fritz is green in a series of steps:

1. Since the base facts indicate that "Fritz croaks" and "Fritz eats flies", the antecedent of rule 1 is satisfied by substituting Fritz for X, and the inference engine concludes:

Fritz is a frog

2. The antecedent of rule 3 is then satisfied by substituting Fritz for X, and the inference engine concludes:

Fritz is green

The name "forward chaining" comes from the fact that the inference engine starts with the data and reasons its way to the answer, as opposed to backward chaining, which works the other way around. In the derivation, the rules are used in the opposite order as compared to backward chaining. In this example, rules 2 and 4 were not used in determining that Fritz is green.

Because the data determines which rules are selected and used, this method is called data-driven, in contrast to goal-driven backward chaining inference. The forward chaining approach is often employed by expert systems, such as CLIPS.

One of the advantages of forward-chaining over backward-chaining is that the reception of new data can trigger new inferences, which makes the engine better suited to dynamic situations in which conditions are likely to change.

Backward Chaining : Backward chaining (or backward reasoning) is an inference method described colloquially as working backward from the goal. It is used in automated theorem provers, inference engines, proof assistants, and other artificial intelligence applications.

In game theory, researchers apply it to (simpler) subgames to find a solution to the game, in a process called *backward induction*. In chess, it is called retrograde analysis, and it is used to generate table bases for chess endgames for computer chess.

Backward chaining is implemented in logic programming by SLD resolution. Both rules are based on the modus ponens inference rule. It is one of the two most commonly used methods of reasoning with inference rules and logical implications – the other is forward chaining. Backward chaining systems usually employ a depth-first search strategy, e.g. Prolog.

Backward chaining starts with a list of goals (or a hypothesis) and works backwards from the consequent to the antecedent to see if any data supports any of these consequents. An inference engine using backward chaining would search the inference rules until it finds one with a consequent (Then clause) that matches a desired goal. If the antecedent (If clause) of that rule is known to be true, then it is added to the list of goals (for one's goal to be confirmed one must also provide data that confirms this new rule).

function FOL-BC-ASK (KB, goals, θ) returns a set of substitutions

inputs : KB, a knowledge base
goals, a list of conjuncts forming a query
 θ , the current substitution, initially the empty substitution

{ }

local variables : ans, a set of substitutions, initially empty

if goals is empty then return { θ }

$q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(\text{goals}))$

for each r in KB where $\text{STANDARDIZE-APART}(r) = (p_1, \dots, p_n \Rightarrow q)$

and $q' \leftarrow \text{UNIFY}(q, q')$ succeeds

$\text{ans} \leftarrow \text{FOL-BC-ASK}(\text{KB}, [p_1, \dots, p_n \mid \text{RESET}(\text{goals})], \text{COMPOSE}(\theta', \theta)) \cup \text{ans}$ return ans

Backward Chaining Algorithm : For example, suppose a new pet, Fritz, is delivered in an opaque box along with two facts about Fritz:

- Fritz croaks
- Fritz eats flies

The goal is to decide whether Fritz is green, based on a rule base containing the following four rules:

1. If X croaks and X eats flies – Then X is a frog
2. If X chirps and X sings – Then X is a canary
3. If X is a frog – Then X is green
4. If X is a canary – Then X is yellow

With backward reasoning, an inference engine can determine whether Fritz is green in four steps. To start, the query is phrased as a goal assertion that is to be proved: "Fritz is green".

1. Fritz is substituted for X in rule 3 to see if its consequent matches the goal, so rule 3 becomes:

If Fritz is a frog – Then Fritz is green

Since the consequent matches the goal ("Fritz is green"), the rules engine now needs to see if the antecedent ("Fritz is a frog") can be proved. The antecedent therefore becomes the new goal:

Fritz is a frog

2. Again substituting Fritz for X, rule 1 becomes:

If Fritz croaks and Fritz eats flies – Then Fritz is a frog

Since the consequent matches the current goal ("Fritz is a frog"), the inference engine now needs to see if the antecedent ("Fritz croaks and eats flies") can be proved. The antecedent therefore becomes the new goal:

Fritz croaks and Fritz eats flies

3. Since this goal is a conjunction of two statements, the inference engine breaks it into two sub-goals, both of which must be proved:

Fritz croaks

Fritz eats flies

4. To prove both of these sub-goals, the inference engine sees that both of these sub-goals were given as initial facts. Therefore, the conjunction is true:

Fritz croaks and Fritz eats flies

therefore the antecedent of rule 1 is true and the consequent must be true:

Fritz is a frog

therefore the antecedent of rule 3 is true and the consequent must be true:

Fritz is green

This derivation therefore allows the inference engine to prove that Fritz is green. Rules 2 and 4 were not used.

Note that the goals always match the affirmed versions of the consequents of implications (and not the negated versions as in modus tollens) and even then, their antecedents are then considered as the new goals (and not the conclusions as in affirming the consequent), which ultimately must match known facts (usually defined as consequents whose antecedents are always true); thus, the inference rule used is modus ponens.

Because the list of goals determines which rules are selected and used, this method is called goal-driven, in contrast to data-driven forward-chaining inference. The backward chaining approach is often employed by expert systems.

Prob.19 Explain Non-monotonic reasoning and discuss various logic associated with it?

[R.T.U. 2015, 2010, Raj. Univ. 2004]

OR

Explain non-monotonic and monotonic reasoning with example.

[R.T.U. 2018]

OR

Explain non monotonic reasoning with suitable example.

[R.T.U. 2017, 2013]

Sol. Non-monotonic Reasoning :

Non-monotonic reasoning, in which the axioms and/or the rules of inference are extended to make it possible to reason with incomplete information. These systems preserve, however, the property that, at any given moment, a statement is either believed to be true, believed to be false or not believed to be either.

Non-monotonic reasoning systems, on the other hand, are designed to be able to solve problems in which all of these properties may be missing.

In order to do this, we must address several key issues, including the following:

1. How can the knowledge base be extended to allow inferences to be made on the basis of lack of knowledge as well as on the presence of it? For example, we would like to

be able
that a
didn't,
not ge
relativ
clear t

•
•

F

on the
reason
system
some p

A

knowl
their r
depend

monot

asserti

absenc

on the

axiom:

no old

In

of som

axiom:

does n

2.

when a

is rem

the add

to be l

conclu

to this

called

justific

and the

a recor

conven

must o

that is l

3.

when t

that co

be base

contrac

in conv

contrac

explici

system

which prohibits someone from having more than one height, then we would not be able to apply the default rule. Thus an explicitly stated value will block the inheritance of a default value, which is exactly what we want.

But now, let's encode the default rule for the height of adult males in general. If we pattern it after the one for baseball players, we get

Adult - Male(x) : height(x, 5-10)
height(x, 5-10)

Unfortunately, this rule does not work as we would like. In particular, if we again assert Pitcher(Three-Finger-Brown), then the resulting theory contains two extensions: one in which our first rule fires and Brown's height is 6-1 and one in which this new rule applies and Brown's height is 5-10. Neither of these extensions is preferred. In order to state that we prefer to get a value from the more specific category, baseball player, we could rewrite the default rule for adult males in general as:

Adult - Male(x) : \neg Baseball-Player(x) \wedge height(x, 5-10)
height(x, 5-10)

This effectively blocks the application of the default knowledge about adult males in the case that more specific information from the class of baseball players is available.

Unfortunately, this approach can become unwieldy as the set of exceptions to the general rule increases. For example, we could end up with a rule like:

Adult - Male(x) : \neg Baseball - Player \wedge
 \neg Midget(x) \wedge \neg Jockey(x) \wedge height(x, 5-10)
height(x, 5-10)

What we have done here is to clutter our knowledge about the general class of adult males with a list of all the known exceptions with respect to height. A clearer approach is to say something like, "Adult males typically have a height of 5-10 unless they are abnormal in some way." We can then associate with other classes the information that they are abnormal in one or another way. So we could write, for example:

$\forall x$: Adult - Male(x) \wedge \neg AB(x, aspect1) \rightarrow height(x, 5-10)

$\forall x$: Baseball - Player(x) \rightarrow AB(x, aspect1)

$\forall x$: Midget(x) \rightarrow AB(x, aspect1)

$\forall x$: Jockey(x) \rightarrow AB(x, aspect1)

Then, if we add the single default rule:

\neg AB(x, y)
 \neg AB(x, y)

we get the desired result.

Sol. Bayesian networks are a type of probabilistic graphical model that uses Bayesian inference for probability computations. Bayesian networks aim to model conditional dependence, and therefore causation, by representing conditional dependence by edges in a directed graph. Through these relationships, one can efficiently conduct inference on the random variables in the graph through the use of factors.

Probability: Before going into exactly what a Bayesian network is, it is first useful to review probability theory. First, remember that the joint probability distribution of random variables A_0, A_1, \dots, A_n , denoted as $P(A_0, A_1, \dots, A_n)$, is equal to $P(A_1 | A_0, \dots, A_n) * P(A_2 | A_3, \dots, A_n) * \dots * P(A_n)$ by the chain rule of probability. We can consider this a factorized representation of the distribution, since it is a product of N factors that are localized probabilities.

$$P\left(\bigcap_{k=1}^n A_k\right) = \prod_{k=1}^n P\left(A_k \mid \bigcap_{j=1}^{k-1} A_j\right)$$

Next, recall that conditional independence between two random variables, A and B , given another random variable, C , is equivalent to satisfying the following property: $P(A, B | C) = P(A | C) * P(B | C)$. In other words, as long as the value of C is known and fixed, A and B are independent.

The Bayesian Network: Using the relationships specified by our Bayesian network, we can obtain a compact, factorized representation of the joint probability distribution by taking advantage of conditional independence.

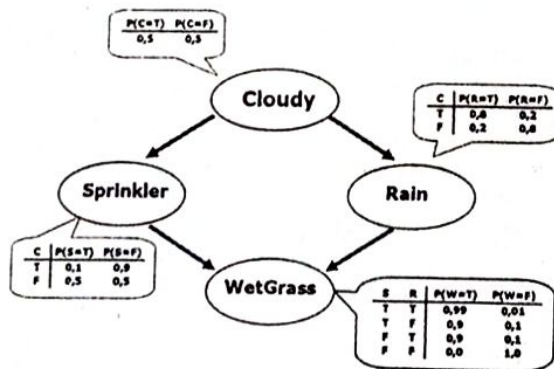


Fig.

A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable. Formally, if an edge (A, B) exists in the graph connecting random variables A and B , it means that $P(B|A)$ is a factor in the joint probability distribution, so we must know $P(B|A)$ for all values of B and A in order to conduct inference. In the above example, since Rain has an edge going into WetGrass, it means that $P(WetGrass|Rain)$ will be a factor, whose

probability values are specified next to the WetGrass node in a conditional probability table

Bayesian networks satisfy the local Markov property, which states that a node is conditionally independent of its non-descendants given its parents. In the above example, this means that $P(\text{Sprinkler} | \text{Cloudy}, \text{Rain}) = P(\text{Sprinkler} | \text{Cloudy})$ since Sprinkler is conditionally independent of its non-descendant, Rain, given Cloudy. After simplification, the joint distribution for a Bayesian network is equal to the product of $P(\text{node} | \text{parents}(\text{node}))$ for all nodes, stated below:

$$P(X_1, \dots, X_n) = \prod_{k=1}^n P(X_k | X_1, \dots, X_{k-1})$$

$$= \prod_{k=1}^n P(X_k | \text{Parents}(X_k))$$

In larger networks, this property allows us to greatly reduce the amount of required computation, since generally, most nodes will have few parents relative to the overall size of the network.

Inference: Inference over a Bayesian network can come in two forms. The first is simply evaluating the joint probability of a particular assignment of values for each variable (or a subset) in the network. For this, we already have a factorized form of the joint distribution, so we simply evaluate that product using the provided conditional probabilities. If we only care about a subset of variables, we will need to marginalize out the ones we are not interested in. In many cases, this may result in underflow, so it is common to take the logarithm of that product, which is equivalent to adding up the individual logarithms of each term in the product.

The second, more interesting inference task, is to find $P(x|e)$, or, to find the probability of some assignment of a subset of the variables (x) given assignments of other variables (our evidence, e). An example of this could be to find $P(\text{Sprinkler}, \text{WetGrass} | \text{Cloudy})$, where $\{\text{Sprinkler}, \text{WetGrass}\}$ is our x , and $\{\text{Cloudy}\}$ is our e . In order to calculate this, we use the fact that $P(x|e) = P(x, e) / P(e) = \alpha P(x, e)$, where α is a normalization constant that we will calculate at the end such that $P(x|e) + P(\neg x|e) = 1$. In order to calculate $P(x, e)$, we must marginalize the joint probability distribution over the variables that do not appear in x or e , which we will denote as Y .

$$P(x|e) = \alpha \sum_{y \in Y} P(x, e, Y)$$

For the given example, we can calculate $P(\text{Sprinkler}, \text{WetGrass} | \text{Cloudy})$ as follows:

We would calculate $P(\neg x | e)$ in the same fashion, just setting the value of the variables in x to false instead of true. Once both $P(x | e)$ and $P(\neg x | e)$ are calculated, we can solve for α , which equals $1 / (P(x | e) + P(\neg x | e))$.

PREVIOUS YEARS QUESTIONS

LEARNING 4

PART-A

Prob.1 What do you mean by learning by chunking?

Sol. Chunking is a process similar in flavour to macro-operators. The idea of chunking comes from the Psychological Literature on memory and problem solving.

Prob.2 Write the methods used for learning.

Sol.

- Memorization
- Direct Instruction
- Analogy
- Induction
- Deduction

Prob.3 Define inductive learning.

Sol. This involves the process of learning by example where a system tries to induce a general rule from a set of observed instances.

Prob.4 What do you mean by Feed forward Neural network?

Sol. This neural network was the first and arguably most simple type of artificial neural network devised. In this network the information moves only in one direction.

Prob.5 What is Fully Recurrent Network?

Sol. This is the basic architecture developed in the 1980s: a network of neuron like units each with a directed connection to every other unit.

PART-B

Prob.6 What do you mean by learning? Explain any one technique which is used in learning? [R.T.U. 2019, 2013]

OR
Explain explanation based learning by taking suitable example. [R.T.U. 2011]

Sol. Learning : It means try to describe exactly what activities we mean and what mechanism could be used to enable us to perform those activities. Learning denotes changes in the system that are adaptive in the sense that enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time.

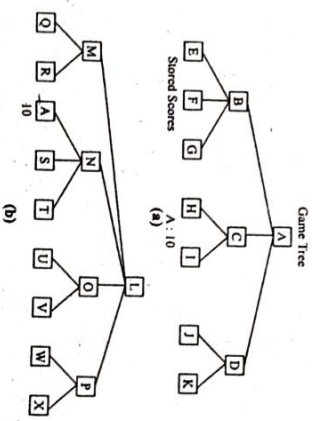


Fig. : Storing Backed-Up Values

Artificial Intelligence

Technique

Explanation Based Learning : EBL is the technique which is used in learning consider a chess player who, as Black, has reached the position shown in Fig. the position is called a "fork" because the white knight attacks both the black king and the black queen. Black must move the king, thereby leaving the queen open to capture. From this single experience, Black is able to learn quite a bit about the fork trap : the idea is that if any piece x attacks both the opponent's king and another piece y, then piece y will be lost. We don't need to see dozens of positive and negative examples of fork positions in order to draw these conclusions. From just one experience, we can learn to avoid this trap in the future and perhaps to use it to our own advantage.

What makes such single-example learning possible? The answer, not surprisingly, is knowledge. The chess player has plenty of domain-specific knowledge that can be brought to bear, including the rules of chess and any previously acquired strategies. That knowledge can be used to identify the critical aspects of the training example. In the case of the fork, we know that the double simultaneous attack is important while the precise position and type of the attacking piece is not.

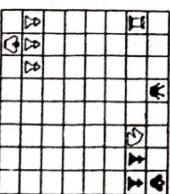


Fig. : A Fork Position in Chess

Much of the recent work in machine learning has moved away from the empirical, data-intensive approach described in the last section toward this more analytical knowledge-intensive approach. A number of independent studies led to the characterization of this approach as explanation-based learning. An EBL system attempts to learn from a single example x by explaining why x is an example of the target concept. The explanation is then generalized and the system's performance is improved through the availability of this knowledge.

Mitchell et al and DeJong and Mooney both describe general frame-works for EBL programs and give general learning algorithms. We can think of EBL programs as accepting the following as input :

- A Training Example: What the learning program "sees" in the world.
- A Goal Concept: A high-level description of what the program is supposed to learn
- An Operationality Criterion: A description of which concepts are usable

- A Domain Theory: A set of rules that describe relationships between objects and actions in a domain.

From this, EBL computes a generalization of the training example that is sufficient to describe the goal concept and also satisfies the operationality criterion.

Prob.7 Explain various techniques used in learning. [R.T.U. 2018]

Sol. Learning techniques available:

1. **Rote learning :** Rote learning is a kind of learning where there is no prior knowledge. When a computer stores a piece of data, it is performing an elementary form of learning. This act of storage presumably allows the program to perform better in the future. Examples of correct behavior are stored and when a new situation arises it is matched with the learnt examples. The values are stored so that they are not re-computed later. One of the earliest game-playing programs is [Samuel, 1963] checkers program.

This program learned to play checkers well enough to beat its creator/designer.

2. **Deductive learning :** Deductive learning works on existing facts and knowledge and deduces new knowledge from the old. This is best illustrated by giving an example. For example, assume:

A=B

B=C

C=A

Then we can deduce with much confidence that:

C=A

Arguably, deductive learning does not generate "new" knowledge at all, it simply memorizes the logical consequences of what is known already. This implies that virtually all mathematical research would not be classified as learning "new" things. However, regardless of whether this is termed as new knowledge or not, it certainly makes the reasoning system more efficient.

3. **Inductive learning :** Inductive learning takes examples and generalizes rather than starting with existing knowledge. For example, having seen many cats, all of which have tails, one might conclude that all cats have tails. This is an unsound step of reasoning but it would be impossible to function without using induction to some extent. In many areas it is an explicit but still it is a useful technique that has been used as the basis of several successful systems. One major subclass of inductive learning is concept learning. This takes examples of a concept and tries to build a general description of the concept. Very often, the examples are described using attribute-value pairs.

techniques can be used in more general program solving programs. The idea is the same: to avoid expensive recomputation. For example, suppose we are faced with the problem of getting to the downtown post office. Our solution may involve getting in car, starting it, and driving along a certain route. Substantial planning may go into choosing the appropriate route, but we need not plan about how to go about starting our car. You are free to treat START-CAR as an atomic action, even though it really consists of several actions: sitting down, adjusting the mirror, inserting the key and turning the key. Sequences of action can be treated as a whole are called macro-operators.

Learning by Chunking : Chunking is a process similar in flavour to macro-operators. The idea of chunking comes from the psychological literature on memory and problem solving. Its computational basis is in production systems. Chunking is a universal learning method, i.e., it can amount for all types of learning in intelligent systems.

The Utility Problem : Once mechanism uses explanation-based learning (EBL), PRODIGY can examine a trace of it shown problem-solving behaviour and try to explain why certain paths failed. The program uses those explanations to formulate control rules that help the problem solver avoid those paths in the future.

D. Learning from Examples : Induction Classification is the process of assigning, to a particular input, the name of a class to which it belongs.

Classification is an important component of many problem-solving tasks. In its simplest form, it is presented as a straightforward recognition task. An example of this is the question "What letter of the alphabet is this?" But often classification is embedded inside another operation. To see how this can happen, consider a problem-solving system that contains the following production rule :

If the current goal is to get from place A to place B and there is a WALL separating the two places.

Then : look for a DOORWAY in the WALL and go through it.

Prob.9 Explain the concept of hopfield neural network with suitable sketch with its applications.

[R.T.U. 2017, 2014]

OR
Explain the concept of hopfield neural network with suitable sketch. What are the application of Hopfield neural network?
[R.T.U. 2011]

Sol. The history of AI is curious. The first problems attacked by AI researchers were problems such as chess and theorem proving, because they were thought to require the essence of intelligence. Vision and language understanding-processes easily mastered by five-year olds - were not thought to be difficult. These days, we have expert chess programs and expert medical diagnosis programs, but no programs that can match the basic perceptual skills of a child. Neural network researchers contend that there is a basic mismatch between standard computer information processing technology and the technology used by the brain.

In addition to these perceptual tasks, AI is just starting to grapple with the fundamental problems of memory and commonsense reasoning. Computers are notorious for their lack of commonsense. Many people believe that common sense derives from our massive store of knowledge and more important, our ability to access relevant knowledge quickly, effortlessly and at the right time.

When we read the description "gray, large, mammal" we automatically think of elephants and their associated features. We access our memories by content. In traditional implementations, access by content involves expensive searching and matching procedures. Massively parallel networks suggest a more efficient method.

Hopfield [1982] introduced a neural network that he proposed as a theory of memory. A hopfield network has the following interesting features :

Distributed Representation : A memory is stored as a pattern of activation across a set of processing elements. Furthermore, memories can be superimposed on one another; different memories are represented by different patterns over the same set of processing elements.

Distributed, Asynchronous Control : Each processing element makes decisions based only on its own local situation. All these local actions add up to a global solution.

Content-Addressable Memory : A number of patterns can be stored in a network. To retrieve a pattern, we need only specify a portion of it. The network automatically finds the closest match.

Fault Tolerance : If a few processing elements misbehave or fail completely, the network will still function properly.

How are these features achieved? A simple Hopfield net is shown in fig., processing elements or units are always in one of two states, active or inactive. In the fig., units colored black are active and units colored white are inactive. Units are connected to each other with weighted, symmetric connections. A positively weighted connection indicates that the two units tend to activate each other. A negative connection allows an active unit to deactivate a neighboring unit.

The network operates as follows. A random unit is chosen. If any of its neighbors are active, the unit computes the sum of the weights on the connections to those active neighbors. If the sum is positive, the unit becomes active, otherwise it becomes inactive. Another random unit is chosen and the process repeats until the network reaches a stable state, i.e., until no more units can change state. This process is called parallel relaxation. If the network starts in the state shown in fig., the unit in the lower left corner will tend to activate the unit above it. This unit, in turn, will attempt to activate the unit above it, but the inhibitory connection from the upper-right unit will foil this attempt and so on.

This network has only four distinct stable states, which are shown in fig. Given any initial state, the network will necessarily settle into one of these four configurations. The network can be thought of as "storing" the patterns in fig. Hopfield's major contribution was to show that given any set of weights and any initial state, his parallel relaxation algorithm would eventually steer the network into a stable state. There can be no divergence oscillation.

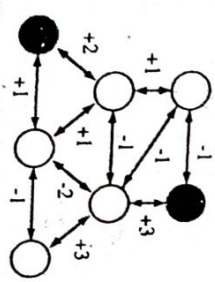


Fig. : A Simple Hopfield Network

The network can be used as a content-addressable memory by setting the activities of the units to correspond to a partial pattern. To retrieve a pattern, we need only supply a portion of it. The network will then settle into the stable state that best matches the partial pattern. An example is shown in fig.

Parallel relaxation is nothing more than search, albeit of a different style than the search described in the early chapters of this book. It is useful to think of the various states of a network as forming a search space, as in fig. A randomly chosen state will transform itself ultimately into one of the local minima, namely the nearest stable state. This is how we get the content-addressable behaviour. We also get error correcting behaviour. Suppose we read the description, "gray, large, fish, eats plankton." We imagine a whale, even though we know that a whale is a mammal, not a fish. Even if the initial state contains inconsistencies, a Hopfield network will settle into the solution that violates the fewest constraints offered by the inputs. Traditional match-and-retrieve procedures are less forgiving.

Now, suppose a unit occasionally fails, say, by becoming active or inactive when it should not. This causes no major problem: surrounding units will quickly set it straight again. It would take the unlikely concerted effort of many errant units to push the network into the wrong stable state. In networks

of thousands of more highly interconnected units, such fault tolerance is even more apparent—units and connections can disappear completely without adversely affecting the overall behavior of the network.

So parallel networks of simple elements can compute interesting things. The next important question is: What is the relationship between the weights on the network's connections and the local minima it settles into? In other words, if the weights encode the knowledge of a particular network, then how is that knowledge acquired? There are several ways to acquire symbolic structures and descriptions. Such acquisition was quite difficult. One feature of connectionist architectures is that their method of representation (namely, real-valued connection weights) lends itself very nicely to automatic learning.

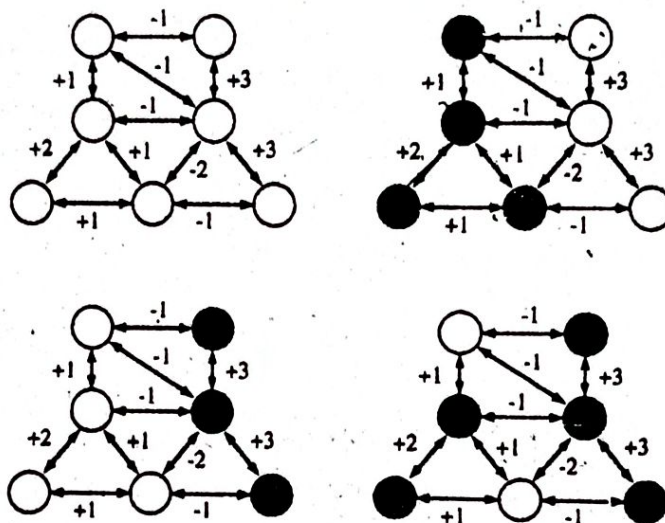


Fig. : The Four Stable States of a Particular Hopfield Net

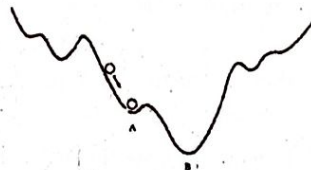


Fig. : A Hopfield Net as a Model of Content-Addressable Memory

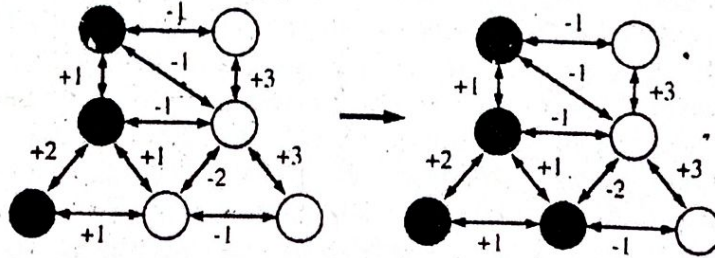


Fig. : A Simplified View of what a Hopfield Net Computes

In the next section, we look closely at learning in several neural network models, including perceptrons, backpropagation networks and Boltzmann machines, a variation of Hopfield networks. After this, we investigate some applications of connectionism. Then we see how networks with feedback can deal with temporal processes and how distributed representations can be made efficient.

PART-C

Prof. 14 Define neural network and explain its application. [R.T.U. 2019, 2013]

OR

What is neural network? Define applications of neural networks. [R.T.U. 2018]

OR

Explain the different types of artificial neural network of architecture. [R.T.U. 2012]

Sol. An Artificial Neural Network is a network of many very simple processors ("units"), each possibly having a (small amount of) local memory. The units are connected by unidirectional communication channels ("connections"), which carry numeric (as opposed to symbolic) data. The units operate only on their local data and on the inputs they receive via the connections.

The design motivation is what distinguishes neural networks from other mathematical techniques: A neural network is a processing device, either an algorithm, or actual hardware, whose design was motivated by the design and functioning of human brains and components thereof.

There are many different types of Neural Networks, each of which has different strengths particular to their applications. The abilities of different networks can be related to their structure, dynamics and learning methods.

1. Feedforward Neural Network : The feedforward neural network was the first and arguably most simple type of artificial neural network devised. In this network the information moves in only one direction — forwards: From the input nodes data goes through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network. Feedforward networks can be constructed from different types of units, e.g. binary McCulloch-Pitts neurons, the simplest example being the perceptron. Continuous neurons, frequently with sigmoidal activation, are used in the context of backpropagation of error.

2. Radial Basis Function (RBF) Network : Radial basis functions are powerful techniques for interpolation in multidimensional space. A RBF is a function which has built into a distance criterion with respect to a center. Radial basis functions have been applied in the area of neural networks where they may be used as a replacement for the sigmoidal hidden layer transfer characteristic in multi-layer perceptrons. RBF networks have two layers of processing: In the first, input is mapped onto each RBF in the 'hidden' layer. The RBF chosen is usually a Gaussian. In regression problems the output layer is then a linear combination of hidden layer

values representing mean predicted output. The interpretation of this output layer value is the same as a regression model in statistics. In classification problems the output layer is typically a sigmoid function of a linear combination of hidden layer values, representing a posterior probability. Performance in both cases is often improved by shrinkage techniques, known as ridge regression in classical statistics and known to correspond to a prior belief in small parameter values (and therefore smooth output functions) in a Bayesian framework.

RBF networks have the advantage of not suffering from local minima in the same way as Multi-Layer Perceptrons. This is because the only parameters that are adjusted in the learning process are the linear mapping from hidden layer to output layer. Linearity ensures that the error surface is quadratic and therefore has a single easily found minimum. In regression problems this can be found in one matrix operation. In classification problems the fixed non-linearity introduced by the sigmoid output function is most efficiently dealt with using iteratively re-weighted least squares.

3. Kohonen Self-Organizing Network : The self-organizing map (SOM) invented by Teuvo Kohonen performs a form of unsupervised learning. A set of artificial neurons learn to map points in an input space to coordinates in an output space. The input space can have different dimensions and topology from the output space and the SOM will attempt to preserve these.

4. Learning Vector Quantization : Learning Vector Quantization (LVQ) can also be interpreted as a neural network architecture. It was suggested by Teuvo Kohonen, originally. In LVQ, prototypical representatives of the classes parameterize, together with an appropriate distance measure, a distance-based classification scheme.

5. Recurrent Neural Network : Contrary to feedforward networks recurrent neural networks (RNNs) are models with bi-directional data flow. While a feedforward network propagates data linearly from input to output, RNNs also propagate data from later processing stages to earlier stages. RNNs can be used as general sequence processors.

6. Fully Recurrent Network : This is the basic architecture developed in the 1980s: a network of neuron-like units, each with a directed connection to every other unit. Each unit has a time-varying real-valued activation. Each connection has a modifiable real-valued weight. Some of the nodes are called input nodes, some output nodes, the rest hidden nodes. Most architectures below are special cases.

For supervised learning in discrete time settings, training sequences of real-valued input vectors become sequences of activations of the input nodes, one input vector at a time. At any given time step, each non-input unit computes its current activation as a nonlinear function of the weighted sum of the activations of all units from which it receives connections. There may be teacher-given target activations for some of the output units at certain time steps. For example,

if the input sequence is a speech signal corresponding to a spoken digit, the final target output at the end of the sequence may be a label classifying the digit. For each sequence, its error is the sum of the deviations of all target signals from the corresponding activations computed by the network. For a training set of numerous sequences, the total error is the sum of the errors of all individual sequences.

Hierarchical RNN

There are many instances of hierarchical RNN whose elements are connected in various ways to decompose hierarchical behaviour into useful subprograms.

7. Stochastic Neural Networks : A stochastic neural network differs from a typical neural network because it introduces random variations into the network. In a probabilistic view of neural networks, such random variations can be viewed as a form of statistical sampling, such as Monte Carlo sampling.

8. Modular Neural Networks : Biological studies have shown that the human brain functions not as a single massive network, but as a collection of small networks. This realization gave birth to the concept of modular neural networks, in which several small networks cooperate or compete to solve problems.

9. Associative Neural Network (ASNN) : The ASNN is an extension of the committee of machines that goes beyond a simple/weighted average of different models. ASNN represents a combination of an ensemble of feedforward neural networks and the k-nearest neighbor technique (kNN). It uses the correlation between ensemble responses as a measure of distance amid the analyzed cases for the kNN. This corrects the bias of the neural network ensemble. An associative neural network has a memory that can coincide with the training set. If new data become available, the network instantly improves its predictive ability and provides data approximation (self-learn the data) without a need to retrain the ensemble. Another important feature of ASNN is the possibility to interpret neural network results by analysis of correlations between data cases in the space of models.

10. Physical Neural Network : A physical neural network includes electrically adjustable resistance material to simulate artificial synapses. Examples include the ADALINE neural network developed by Bernard Widrow in 1960s and the memristor based neural network developed by Greg Snider of HP Labs in 2008.

The applications features here are :

- Coevolution of Neural Networks for Control of Pursuit & Evasion
- Learning the Distribution of Object Trajectories for Event Recognition
- Radiosity for Virtual Reality Systems

- Autonomous Walker & Swimming Eel
- Robocup: Robot World Cup
- Using HMM's for Audio-to-Visual Conversion
- Artificial Life: Galapagos
- Speech reading (Lipreading)
- Detection and Tracking of Moving Targets
- Real-time Target Identification for Security Applications
- Facial Animation
- Behavioural Animation and Evolution of Behaviour
- A Three Layer Feedforward Neural Network
- Artificial Life for Graphics, Animation, Multimedia and Virtual Reality: Sigraph '95 Showcase
- Creatures : The World Most Advanced Artificial Life!
- Framsticks Artificial Life

Prob.15 Discuss the single layer perceptron model of neural network. What are the limitations of this model? [R.T.U. 2016]

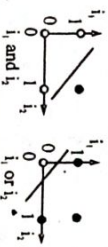
OR
Explain single layer perceptron model of the neural network. What are its features? [R.T.U. 2019, 2013]

OR

Explain the single layer perceptron model of neural network. What are limitations of this model? [R.T.U. 2011]

Sol. Single Layer Perceptron Network

Single-layer neural networks (perceptron networks) are networks in which the output unit is independent of the others, each weight effects only one output. The simplest kind of neural network is a single-layer perceptron network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. In this way it can be considered the simplest kind of feed-forward network. The sum of the products of each node and the inputs is calculated in each node and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1). Neurons with this kind of activation function are also called Artificial neurons or linear threshold units. In the literature the term perceptron often refers to networks consisting of just one of these units. Using perceptron networks it is possible to achieve linear separability functions like the diagrams shown below (assuming we have a network with 2 inputs and 1 output)



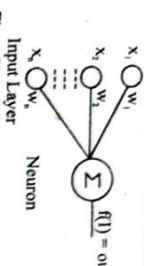
It can be seen that this is equivalent to the AND / OR logic gates, shown below.

AND logic gate

I_1	I_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR logic gate

I_1	I_2	Output
0	0	0
0	1	1
1	0	1
1	1	1



The input layer is the first set of nodes that bring in activation potentials from other neurons. We have n of them in the diagram above. They are equivalent to dendrites in the biological representation.

The processing unit sums the inputs it receives, then applies an activation function to the sum of the input signals. This sum forms the basis of whether this neuron should fire positive or negative to the next neuron/receptor/effector in the line.

The output line then transmits the result to other neurons. This is the part where the synaptic cleft would appear between $f(I)$ and X_n .

The M&P model was the first to treat the human brain as a computational organism. The key concept of it was to assign each input a 'weight' that would tell the neuron to what extent it should 'take notice' of the input.

The model then used a threshold function to determine whether the neuron should fire. If the weighted sum of the inputs was greater than this threshold value the neuron would fire positively (1), otherwise negatively (0).

$$I = \sum x_i w_i > T$$

The neuron will fire positive for sums $> T$ and negative for sums $< T$. (we may decide what we want to happen when the sum is equal to T but here we have said only for values larger than T will we fire positively).

Limitations

- It has no hidden layers so cannot solve linearly inseparable problems (see XOR).
- It only allows binary output 0 or 1
- No feedback is possible.
- The XOR is not linear separable.
- It is impossible to separate the classes I_1 and I_2 with only one line.
- Single-unit perceptrons are only capable of learning linearly separable patterns;

• It is often believed that they also conjectured (incorrectly) that a similar result would hold for a multi-layer perceptron network.

• A single threshold unit is quite limited in its computational power. Refer to Prob.14.

Prob.16 How "learning by example" is different from "learning by taking advice"? Explain it giving suitable example. [R.T.U. 2017, 2014]

OR
Differentiate the "Learning by taking advice" and "Learning by example" with an example. [R.T.U. 2019, 2013, 2011]

Sol. Learning by Taking Advice

The idea of advice taking in AI based learning was proposed as early as 1958 (McCarthy). However, very few attempts were made in creating such systems until the late 1970s.

There are two basic approaches to advice taking:

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

(i) Automated Advice Taking

The following steps summarize this method:

- Request** : This can be simple question asking about general advice or more complicated by identifying shortcomings in the knowledge base and asking for a remedy.
- Interpret** : Translate the advice into an internal representation.
- Operationalize** : Translated advice may still not be usable so this stage seeks to provide a representation that can be used by the performance element.
- Integrate** : When knowledge is added to the knowledge base care must be taken so that bad side-effects are avoided.

E.g. Introduction of redundancy and contradictions.

(e) **Evaluate** : The system must assess the new knowledge for errors, contradictions etc.

The steps can be iterated.

(ii) **Knowledge Base Maintenance**
Instead of automating the five steps above, many researchers have instead assembled tools that aid the development and maintenance of the knowledge base. Many have concentrated on :

- Providing intelligent editors and flexible representation languages for integrating new knowledge
- Providing debugging tools for evaluating, finding contradictions and redundancy in the existing knowledge base.

Example Learning System - FOO
Learning the game of hearts
FOO (First Operational Operationalizer) tries to convert high level advice (principles, problems, methods) into effective executable (LISP) procedures.

Hearts :

- Game played as a series of tricks.
- One player - who has the lead - plays a card.
- Other players follow in turn and play a card.
- The player must follow suit.
- - If he cannot, he play any of his cards.
- The player who plays the highest value card wins the trick and the lead.
- The winning player takes the cards played in the trick.

- The aim is to avoid taking points. Each heart counts as one point the queen of spades is worth 13 points.
- The winner is the person that after all tricks have been played has the lowest points score.
- Hearts is a game of partial information with no known algorithm for winning.
- Although the possible situations are numerous general advice can be given such as :
- Avoid taking points.
- Do not lead a high card in suit in which an opponent is void.
- If an opponent has the queen of spades try to flush it.

In order to receive advice a human must convert into a FOO representation (LISP clause)

(avoid (take-points me) (trick))

FOO operationalizes the advice by translating it into expressions it can use in the game. It can UNFOLD avoid and then trick to give :

(achieve (not (during (scenario (each pl (players) (play-card pl)) (take-trick (trick-winner)) (take-points me))))))

However, the advice is still not operational since it depends on the outcome of trick which is generally not known. Therefore FOO uses case analysis (on the *during* expression) to determine which steps could case one to take points. Step 1 is ruled out and step 2's take-points is UNFOLDED :

(achieve (not (during (scenario (each pl (players) (play-card pl)) (take-trick (trick-winner)) (take-points me))))))

Once more remove inconsistencies from G and then generalize S:

$$G = \{(x_1 = x_2 = x_3 = x_4 = x_5 = \text{samcsuit}, x_1, x_2, x_3, x_4, x_5)\}$$

$$G = \{(x_1 = x_2 = x_3 = x_4 = x_5 = \text{samcsuit}, x_2, x_3, x_4, x_5)\}$$

- We can continue generalizing and specializing.
- We have taken a few big jumps in the flow of specializing/generalizing in this example. Many more training steps usually required to reach this conclusion.
- It might be hard to spot trend of same suit, etc.

(iii) Decision Trees

Quinlan in his ID3 system (1986) introduced the idea of decision trees. ID3 is a program that can build trees automatically from given positive and negative instances. Basically each leaf of a decision tree asserts a positive or negative concept. To classify a particular input we start at the top and follow assertions down until we reach an answer (fig).

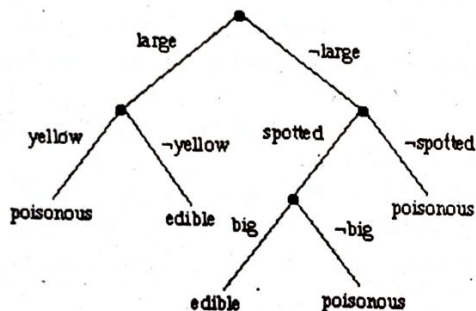


Fig. : Edible Mushroom Decision Tree

Building Decision trees

- ID3 uses an iterative method.
- Simple trees preferred as more accurate classification is afforded.
- A random choice of samples from training set chosen for initial assembly of tree - the window subset.
- Other training examples used to test tree.
- If all examples classified correctly stop.
- Otherwise add a number of training examples to window and start again.

Adding New Nodes

When assembling the tree we need to choose when to add a new node:

- Some attributes will yield more information than others.
- Adding a new node might be useless in the overall classification process.
- Sometimes attributes will separate training instances into subsets whose members share a common label. Here branching can be terminated and a leaf node assigned for the whole subset.

Decision Tree Advantages

- Quicker than version spaces when concept space is large.
- Disjunction easier.

Disadvantages

Representation not natural to humans - a decision tree may find it hard to explain its classification.

Prob.17 Write a short note on Winston learning program.
[R.T.U. 2016]

OR

Discuss winston's learning program? [R.T.U. 2015]

Sol. Winston's doctoral thesis at MIT entitled "Learning Structural Descriptions from Examples" (1970) was a major step towards a clarification of how concepts involving complex structural relationships might be learned.

His program is presented with line drawings of scenes containing children's toy blocks, such as bricks, cubes, pyramids, and wedges. The program forms descriptive networks for these scenes, which shows the properties and relationships of the objects appearing in them. Using these structural descriptions, the program can learn structural concepts such as "pedestal", "arch" or "arcade" on the basis of examples and counter examples of the concepts.

While the content of the scenes is severely restricted, the methods employed in the program seem quite general, or at least readily generalizable to more realistic concept-formation tasks (consider "chair", "table", "house", etc.) Certainly this type of structural learning is a long way off from learning by parameter adjustment, which has been the dominant paradigm in pattern recognition research (Uhr&Vossler's program and some others excepted).

Winston's program uses Guzman's algorithm to determine the bodies in a scene; it then determines which edges belong to which object and fills in partially occluded edges. Then it infers the types of objects (brick, wedge, etc.) from the shapes and adjacency relationships of the viable faces. The sizes and orientation are then readily available.

Winston (1975) described a Blocks World Learning program. This program operated in a simple blocks domain. The goal is to construct representation of the definition of concepts in the blocks domain.

Example: Concepts such a "house".

- Start with input, a line drawing of a blocks world structure.
- It learned Concepts House, Tent, Arch as : brick (rectangular block) with a wedge (triangular block)

Simon has proposed that learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time.

Learning cover a wide range of phenomena.

(a) **Skill Refinement** : People get better at many tasks simply by practicing. The more you ride a bicycle or play tennis, the better you get.

(b) **Knowledge Acquisition** : Knowledge is generally acquired through experience.

Method used for learning are

- Memorization (Rote Learning)
- Direct Instruction (By being told)
- Analogy
- Induction
- Deduction.

(ii) **Learning by Example** : Refer to Prob.16.

(iii) **Explanation Based Learning (EBL)** : Refer to Prob.10.

Prob.19 Write detail note on SVM. Also explain its application.

Sol. Support Vector Machine (SVM) was first heard in 1992, introduced by Boser, Guyon, and Vapnik in COLT-92. Support Vector Machines (SVMs) are a set of related supervised learning methods used for classification and regression. They belong to a family of generalized linear classifiers. In other terms, Support Vector Machine (SVM) is a classification and regression prediction tool that uses machine learning theory to maximize predictive accuracy while automatically avoiding over-fit to the data. Support Vector Machines can be defined as systems which use hypothesis space of a linear functions in a high dimensional feature space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory. Support Vector Machine was initially popular with the NIPS community and now is an active part of the machine learning research around the world. SVM becomes famous when, using pixel maps as input; it gives accuracy comparable to sophisticated neural networks with elaborated features in a handwriting recognition task. It is also being used for many applications, such as handwriting analysis, face analysis and so forth, especially for pattern classification and regression based applications. The foundations of Support Vector Machines (SVM) have been developed by Vapnik and gained popularity due to many promising features such as better empirical performance. The formulation uses the Structural Risk Minimization (SRM),

principle, which has been shown to be superior, to traditional Empirical Risk Minimization (ERM) principle, used by conventional neural networks. SRM minimizes an upper bound on the expected risk, where as ERM minimizes the error on the training data. It is this difference which equips SVM with a greater ability to generalize, which is the goal in statistical learning. SVMs were developed to solve the classification problem, but recently they have been extended to solve regression problems.

Introduction to SVM: Why SVM?

Firstly working with neural networks for supervised and unsupervised learning showed good results while used for such learning applications. MLP's uses feed forward and recurrent networks. Multilayer perceptron (MLP) properties include universal approximation of continuous nonlinear functions and include learning with input-output patterns and also involve advanced network architectures with multiple inputs and outputs.

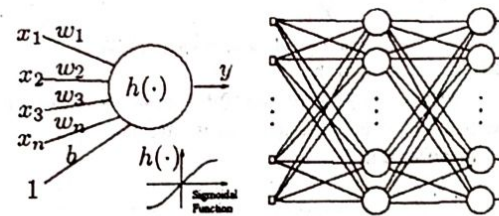


Fig. : (a) Sample Neural Network (b) Multilayer Perceptron.

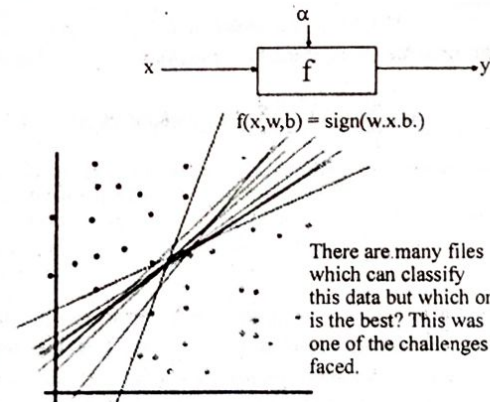


Fig.

There can be some issues noticed. Some of them are having many local minima and also finding how many neurons might be needed for a task is another issue which determines

whether optimality of that NN is reached. Another thing to note is that even if the neural network solutions used tends to converge, this may not result in a unique solution. Now let us look at another example where we plot the data and try to classify it and we see that there are many hyper planes which can classify it.

From above illustration, there are many linear classifiers (hyper planes) that separate the data. However only one of these achieves maximum separation. The reason we need it is because if we use a helper plane to classify, it might end up closer to one set of datasets compared to other and we do not want this to happen and thus we see that the concept of maximum margin classifier or hyper plane as an apparent solution.

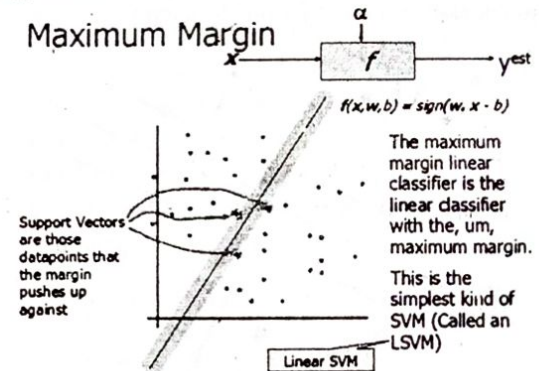


Fig. : Illustration of Linear SVM

Expression for Maximum margin is given as :

$$\text{margin} = \arg \min_{x \in D} d(x) = \arg \min_{x \in D} \frac{|x \cdot w + b|}{\sqrt{\sum_{i=1}^d w_i^2}}$$

The above illustration is the maximum linear classifier with the maximum range. In this context it is an example of a simple linear SVM classifier. There are some good explanations which include better empirical performance. Another reason is that even if we've made a small error in the location of the boundary this gives us least chance of causing a misclassification. The other advantage would be avoiding local minima and better classification. Now we try to express the SVM mathematically to present a linear SVM. The goals of SVM are separating the data with hyper plane and extend this to nonlinear boundaries using kernel trick. For calculating the SVM we see that the goal is to correctly classify all the data. For mathematical calculations we have,

- (a) If $Y_i = +1$; $w \cdot x_i + b \geq 1$
 (b) If $Y_i = -1$; $w \cdot x_i + b \leq -1$
 (c) For all i ; $y_i(w \cdot x_i + b) \geq 1$

In this equation x is a vector point and w is weight and is also a vector. So to separate the data (a) should always be greater than zero. Among all possible hyper planes, SVM selects the one where the distance of hyper plane is as large as possible. If the training data is good and every test vector is located in radius r from training vector. Now if the chosen hyper plane is located at the farthest possible from the data. This desired hyper plane which maximizes the margin also bisects the lines between closest points on convex hull of the two datasets. Thus we have (a), (b) and (c).

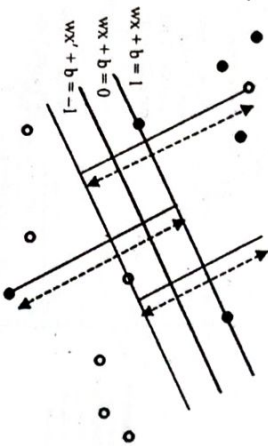


Fig. : Representation of Hyper planes

Distance of closest point on hyper plane to origin can be found by maximizing the x as x is on the hyper plane. Similarly for the other side points we have a similar scenario. Thus solving and subtracting the two distances we get the summed distance from the separating hyper plane to nearest points. Maximum Margin $M = 2 / \|w\|$

Now maximizing the margin is same as minimum. Now we have a quadratic optimization problem and we need to solve for w and b . To solve this we need to optimize the quadratic function with linear constraints. The solution involves constructing a dual problem and where a Lagrange's multiplier a_i is associated. We need to find w and b such that $\phi(w) = 1/2 \|w\|^2$ is minimized.

And for all $\{(x_i, y_i) : y_i (w \cdot x_i + b) \geq 1\}$.

Now solving: we get that $w = \sum a_i \cdot x_i$; $b = y_k - w \cdot x_k$ for any x_k such that $a_k \neq 0$.

B.Tech. (IT Sem J C.S. Solved Papers)
 Now the classifying function will have the following form:

$$f(x) = \sum a_i y_i x_i \cdot x + b$$

- denotes +1
- denotes -1

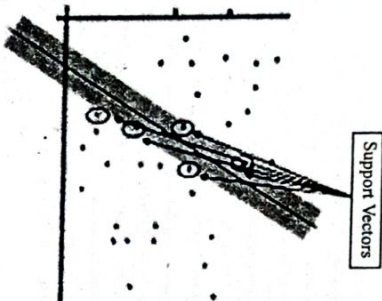


Fig. : Representation of Support Vectors

SVM Representation
 In this we present the QP formulation for SVM classification. This is a simple representation only.

$$\min_{\xi_i} \|f\|_k^2 + C \sum_{i=1}^l \xi_i$$

$$y_i f(x_i) \geq 1 - \xi_i, \text{ for all } i; \xi_i \geq 0$$

SVM classification, Dual Formulation :

$$\min_{a_i} \sum_{i=1}^l a_i - \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l a_i a_j y_i y_j K(x_i, x_j)$$

$$0 \leq a_i \leq C, \text{ for all } i;$$

$$\sum_{i=1}^l a_i y_i = 0$$

Variables ξ_i are called slack variables and they measure the error made at point (x_i, y_i) . Training SVM becomes quite challenging when the number of training points is large. A number of methods for fast SVM training have been proposed.

Applications of SVM : SVM has been found to be successful when used for pattern classification problems. Applying the Support Vector approach to a particular practical problem involves resolving a number of questions based on

the problem definition and the design involved with it. One of the major challenges is that of choosing an appropriate kernel for the given application. There are standard choices such as a Gaussian or polynomial kernel that are the default options, but if these prove ineffective or if the inputs are discrete structures more elaborate kernels will be needed. By implicitly defining a feature space, the kernel provides the description language used by the machine for viewing the data. Once the choice of kernel and optimization criterion has been made the key components of the system are in place. Let's look at some examples.

The task of text categorization is the classification of natural text documents into a fixed number of predefined categories based on their content. Since a document can be assigned to more than one category this is not a multi-class classification problem, but can be viewed as a series of binary classification problems, one for each category. One of the standard representations of text for the purposes of information retrieval provides an ideal feature mapping for constructing a Mercer kernel. Indeed, the kernels somehow incorporate a similarity measure between instances, and it is reasonable to assume that experts working in the specific

application domain have already identified valid similarity measures, particularly in areas such as information retrieval and generative models.

Traditional classification approaches perform poorly when working directly because of the high dimensionality of the data but Support Vector Machines can avoid the pitfalls of very high dimensional representations. A very similar approach to the techniques described for text categorization can also be used for the task of image classification, and as in that case linear hard margin machines are frequently able to generalize well. The first real-world task on which Support Vector Machines were tested was the problem of handwritten character recognition. Furthermore, multi-class SVMs have been tested on these data. It is interesting not only to compare SVMs with other classifiers, but also to compare different SVMs amongst themselves. They turn out to have approximately the same performance, and furthermore to share most of their support vectors, independently of the chosen kernel. The fact that SVM can perform as well as these systems without including any detailed prior knowledge is certainly remarkable.

purpose, they are very useful and will continue to be used. Robotics is the art, knowledge base and the know how of designing, applying and using robots in the human endeavors. Robotics systems consist of not just robots, but also other devices and systems that are used together with the robots to perform the necessary tasks.

Laws of Robotics : Issac Asimov conceived the robots as humanoids, devoid of feelings, and used them in a number of stories. His robots were well-designed, fail-safe machines, whose brains were programmed by human beings. Anticipating the dangers and havoc such a device could cause, he postulated rules for their ethical conduct. Robots were required to perform according to three principles known as "three laws of Robotics", which are as valid for real robots as they were for Asimov's robots, Issac Asimov proposed three laws of robotics and he later added a 'zeroth law' :

1. **First Law :** A robot should not injure a human being or, through inaction, allow a human being to come to harm, unless this would violate a higher order law.
2. **Second Law :** A robot must obey orders given by humans except when that conflicts with the First Law.
3. **Third Law :** A robot must protect its own existence unless that conflicts with the First or Second Law.
4. **Zero Law :** A robot may not injure humanity, or, through in action, allow humanity to come to harm.

These are very general laws and apply even to other machines and appliances. They are always taken care of in any robots design.

Prob.7 What is block world problem? Explain with an example. [R.T.U. 2013, 2011]

OR

Explain block world problem in robotics.

[R.T.U. 2018]

Sol. Example Domain : The Blocks World

The techniques we are about to discuss can be applied in a wide variety of task domains and they have been. But to make it easy to compare the variety of methods we consider, we should find it useful to look at all of them in a single domain that is complex enough that the need for rich of the mechanisms is apparent yet simple enough that easy-to-follow examples can be found. The blocks world is such a domain. There is a flat surface on which blocks can be placed. There are a number of square blocks, all the same size. They can be stacked one upon another. There is a robot arm that can manipulate the blocks. The actions it can perform include:

- **UNSTACK(A, B) :** Pick up block A from its current position on block B. The arm must be empty and block A must have no blocks on top of it.

- **STACK(A, B) :** Place block A on block B. The arm must already be holding A and the surface of B must be clear.
- **PICKUP(A) :** Pick up block A from the table and hold it. The arm must be empty and there must be nothing on top of block A.
- **PUTDOWN(A) :** Put block A down on the table. The arm must have been holding block A.

Notice that in the world we have described, the robot arm can hold only one block at a time. Also, since all blocks are the same size, each block can have at most one other block directly on top of it.

In order to specify both the conditions under which an operation may be performed and the results of performing it, we need to use the following predicates:

- **ON(A, B)**-Block A is on block B.
- **ONTABLE(A)**-Block A is on the table.
- **CLEAR(A)**-There is nothing on top of block A.
- **HOLDING(A)**-The arm is holding block A.
- **ARMEMPTY**-The arm is holding nothing.

Various logical statements are true in this blocks world.

For example,

$[\exists x : \text{HOLDING}(x) \rightarrow \neg \text{ARMEMPTY}]$

$\forall x : \text{ONTABLE}(x) \rightarrow \neg \exists y : \text{ON}(x, y)$

$\forall x : [\neg \exists y : \text{ON}(x, y)] \rightarrow \text{CLEAR}(x)$

The first of these statements says simply that if the arm is holding anything, then it is not empty. The second says that if a block is on the table, then it is not also on another block. The third says that any block with no blocks on it is clear.

Prob.8 Explain the origin and history of Robotics in detail. [R.T.U. 2016]

Sol. Origin and History of Robotics: The term Robotics deals with the mixture of electronics, machine design, and computer programming. In brief, "Robotics" is the study of robot designs and applications. For past three years, Robotics has become an emerging technology and futuristic engineering branch all over the world.

The name Robotics came from the word 'Robot'. A Robot is a machine which can do work by itself without any human assistance. In olden days, robots were only used in dramas, and it was a dream for the real use. As expected, now the robots are highly developed and fully automated for performing a work with maximized accuracy and consistent.

The Problem: No natural language program can be complete because new words, expressions and meanings can be generated quite freely:

I'll fax it to you.

The Good Side: Language can evolve as the experiences that we want to communicate about evolve.

The Problem: There are lots of ways to say the same thing:

Mary was born on October 11.

Mary's birthday is October 11.

The Good Side: When you know a lot, facts imply each other. Language is intended to be used by agents who know a lot.

Prob.10 Write various issues or difficulties involved in Natural Language Processing.

Sol. A very critical problem in natural languages is the ambiguity, which is their inherent weakness, because ambiguity refers to more than one meaning. In the theory of computational linguistics, several types of ambiguities are defined. Here we present only those types of ambiguities, which are normally found in the practical AI based systems:

(i) **Lexical ambiguity:** When one word can have several different meanings, the resulting ambiguity is called lexical. To resolve this ambiguity additional knowledge regarding words is used according to context. For example, consider the following sentences:

- (a) Withdraw some money from the bank.
- (b) Do not go near the bank of the river.

Here, word bank has two entirely different meanings in the context of its use.

(ii) **Syntactic ambiguity:** Sometimes, sentences can be parsed in more than one way, that is, phrases can be put together differently. This ambiguity is resolved by using common sense knowledge about the task. For example, consider the following sentences:

- (a) I saw the boy with the telescope,
- (b) I saw the boy with the telescope.

These two sentences could have entirely different meanings depending upon the interpreter. The first sentence can be referred as 'you saw the boy using a telescope' while the second sentence can be referred as 'you saw the boy having a telescope with him'.

(iii) **Referential ambiguity:** The use of pronoun and other anaphora can cause referential ambiguity. The anaphora are replacement words that are used in place of noun in later part of discourse. The discourse is a term, which indicates multiple coherent sentences. Process for solving referential ambiguity involves complex reasoning on the part of both

speaker and the hearer. The contextual knowledge, knowledge regarding the belief system of the speaker and the hearer, common sense knowledge about the particular task are used to resolve such ambiguity. For example, consider the following sentence:

"Abhishek saw a beautiful Chevrolet Spark at the showroom. He showed it to Amitabh. He bought it".

In this example, 'he' can refer to 'Abhishek' and 'Amitabh' and 'it' can refer to 'Chevrolet Spark' or 'showroom'.

(iv) **Pragmatic ambiguity:** This ambiguity underlies in meaning of the sentence. This arises because of different intentions of the speaker. For example, if an employee reaches office at 9.30 a.m. (whereas office starts at 9.00 a.m.) and the boss asks him "what is the time?", the answer might be 9.30 a.m., but the intention of the boss is to make him realize that he was late. To resolve this ambiguity, the common sense knowledge and contextual knowledge is required.

PART-C

Prob.11 Explain various steps used in natural language processing. Also discuss in detail.

(i) Syntactic processing

(ii) Semantic analysis

[R.T.U. 2016]

OR

What is natural language processing? Explain with example.

[R.T.U. 2019, 2018, 2017, 2013]

OR

Explain the following in detail:

(1) Syntactic processing

(2) Semantic processing

[R.T.U. 2017, 2015]

OR

What are the steps in natural language processing? List and explain them briefly.

[R.T.U. 2015, 2012]

OR

What do you mean by natural language processing? Explain in brief.

[R.T.U. 2014]

OR

What is morphological analysis? Explain with example.

[R.T.U. 2013]

Sol. Natural Language Processing : It is usually shortened as NLP, is a branch of artificial intelligence that deals with

the interaction between computers and humans using the natural language. NLP process has following steps:

1. Morphological Analysis: Individual words are analyzed into their components, and nonword tokens, such as punctuation, are separated from the words.

Morphological analysis must do the following things:

- Pull apart the word "Bill's" into the proper noun "Bill" and the possessive suffix "s".
- Recognize the sequence ".init" as a file extension that is functioning as an adjective in the sentence.

In addition, this process will usually assign syntactic categories to all the words in the sentence. This is usually done now because interpretations for affixes (prefixes and suffixes) may depend on the syntactic category of the complete word. For example, consider the word "prints" is either a plural noun (with the "-s" marking plural) or a third person singular verb (as in "the prints"), in which case the "-s" indicates both singular and third person. If this step is done now, then in our example, there will be ambiguity since "want", "print", and "file" can all function as more than one syntactic category.

2. Syntactic Analysis: Linear sequences of words are transformed into structures that show how the words relate to each other. Some word sequences may be rejected if they violate the language's rules for how words may be combined. For example, an English syntactic analyzer would reject the sentence "Boy the go the to store."

Syntactic analysis must exploit the results of morphological analysis to build a structural description of the sentence. The goal of this process, called parsing, is to convert the flat list of words that forms the sentence into a structure that defines the units that are represented by that flat list. For our example sentence, the result of parsing is shown in Fig. 1. The details of this representation are not particularly significant. What is important here is that a flat sentence has been converted into a hierarchical structure and that structure has been designed to correspond to sentence units (such as noun phrases) that will correspond to meaning units when semantic analysis is performed. One useful thing we have done here, although not all syntactic systems do, is create a set of entities we call reference markers. They are shown in parentheses in the parse tree. Each one corresponds to some parentheses in the parse tree. Each one corresponds to some entity that has been mentioned in the sentence. These reference markers are useful later since they provide a place in which to accumulate information about the entities as we get it. Thus although we have not tried to do semantic analysis (i.e., assign meaning) at this point, we have designed our

syntactic analysis process so that it will find constituents to which meaning can be assigned.

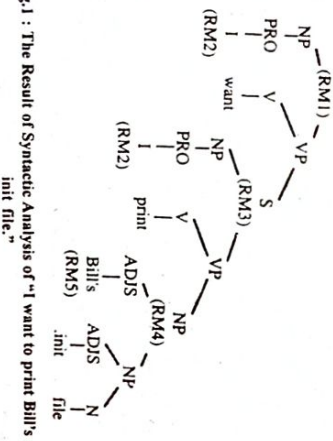


Fig. 1: The Result of Syntactic Analysis of "I want to print Bill's init file."

3. Semantic Analysis: The structures created by the syntactic analyzer are assigned meanings. In other words, a mapping is made between the syntactic structures and objects in the task domain. Structures for which no such mapping is possible may be rejected. For example, in most universes, the sentence "Colorless green ideas sleep furiously" would be rejected as semantically anomalous.

- Semantic analysis must do two important things:
- It must map individual words into appropriate objects in the knowledge base or database.
- It must create the correct structures to correspond to the way the meanings of the individual words combine with each other.

For this example, suppose that we have a frame-based knowledge base that contains the units shown in Fig. 2. Then we can generate a partial meaning, with respect to that knowledge base, as shown in Fig. 3. Reference marker RM1 corresponds to the top-level event of the sentence. It is a wanting event in which the speaker (denoted by "I") wants a printing event to occur in which the same speaker prints a file whose extension is ".init" and whose owner is Bill.

4. Discourse Integration: The meaning of an individual sentence may depend on the sentences that precede it and may influence the meanings of the sentences that follow it. For example, the word "it" in the sentence, "John wanted it," depends on the prior discourse context, while the word "John" may influence the meaning of later sentences (such as, "He always had.")

At this point, we have figured out what kinds of things this sentence is about. But we do not yet know which specific individuals are being referred to. Specifically, we do not know

to whom the pronoun "I" or the proper noun "Bill" refers. To pin down these references requires an appeal to a model of the current discourse context, from which we can learn that the current user (who typed the word "I") is User068 and that the only person named "Bill" about whom we could be talking is User073.

User	isa: Person	must be <string>
User068	* login-name	
instance:	User	
login-name:	Susan-Black	
User073	instance:	User
login-name:	Bill-Smith	
FI	instance:	File-Struct
name:	stuff	
extension:	.init	
owner:	User073	
in-directory:	/wsmith/	
File-Struct	instance:	Information-Object
isa:	Information-Object	
Printing	isa:	Physical-Event
* agent:	must be <animate>	
* object:	must be <state or event>	
Wanting	isa:	Mental-Event
* agent:	must be <animate>	
* object:	must be <animate or program>	
Commanding	isa:	Mental-Event
* agent:	must be <animate>	
* performer:	must be <animate or program>	
* object:	must be <event>	
This-System	instance:	Program
instance:	Program	

Fig. 2: A Knowledge Base Fragment

Once the correct referent for Bill is known, we can also determine exactly which file is being referred to: FI is the only file with the extension ".init" that is owned by Bill.

RM1	instance:	(the whole sentence)
agent:	Wanting	
RM2	agent:	(1)
object:	RM3	(a printing event)
RM2	object:	(1)
RM3	instance:	(a printing event)
agent:	Printing	
RM4	agent:	(1)
object:	RM4	(Bill's init file)
instance:	File-Struct	
extension:	.init	
owner:	RM5	(Bill)
instance:	Person	
first-name:	Bill	

Fig. 3: A Partial Meaning for a Sentence

5. Pragmatic Analysis: The structure representing what was said is reinterpreted to determine what was actually meant. For example, the sentence "Tjogou know what time it is?" should be interpreted as a request to be told the time.

Specifically, to make the overall language understanding problem tractable, it will help if we distinguish between the following two ways of decomposing a program:

- The processes and the knowledge required to perform the task
- The global control structure that is imposed on those processes

We now have a complete description, in the terms provided by our knowledge base, of what was said. The final step toward effective understanding is to decide what to do as a result. One possible thing to do is to record what was said as a fact and be done with it. For some sentences, whose intended effect is clearly declarative, that is precisely the correct thing to do. But for other sentences, including this one, the intended effect is different. We can discover this intended effect by applying a set of rules that characterize cooperative dialogues. In this example, we use the fact that when the user claims to want something that the system is capable of performing, then the system should go ahead and do it.

The final step in pragmatic processing is to translate, when necessary, from the knowledge-based representation to a command to be executed by the system. In this case, this step is necessary, and we see that the final result of the understanding process is

lpr /wsmith/stuff.init

Example Natural Language Processing: In natural language processing, the same expression means different things in different context.

Where's the water?

Now, in a chemistry lab, it must be pure
When we are thirsty, it must be potable

Dealing with a leaky roof, it can be filthy

Example of Morphological Analysis : Suppose we have an English interface to an operating system and the following sentence is typed:

I want to print Bill's init file.

Then, morphological analysis must do the following things:

1. Pull apart the word "Bill's" into the proper noun "Bill".
2. Recognize the sequence ".init" as a file extension.

Prq. 12 Explain expert system in detail with suitable example. Also explain its components. [R.T.U. 2016]

OR

Explain with neat diagram the architecture of expert system and mention its features. [R.T.U. 2016]

OR

Explain expert system with a suitable example. [R.T.U. 2013]

OR

What is the expert system? With block diagram, explain each module of it. [R.T.U. 2011]

OR

What do you understand by an expert system? Explain its components. [R.T.U. 2011]

Sol. Expert system : An expert system is a set of programs that manipulate encoded knowledge to solve problems in a specialized domain that normally requires human expertise. An expert system's knowledge is obtained from expert sources and coded in a form suitable for the system to use in its inference or reasoning processes. The expert knowledge must be obtained from specialists or other sources of expertise, such as texts, journal articles and data bases. This type of knowledge usually requires much training and experience in some specialized field such as medicine, geology, system configuration or engineering design. Once a sufficient body of expert knowledge has been acquired it must be encoded in some form loaded into a knowledge base, then tested, and refined continually throughout the life of the system.

Characteristics and Features of Expert Systems

Expert systems differ from conventional computer systems in several important ways:

1. Expert systems use knowledge rather than data to control the solution process. Much of the knowledge used is heuristic in nature rather than algorithmic.

2. The knowledge is encoded and maintained as an entity separate from the control program. As such, it is not compiled together with the control program and itself. This permits the incremental addition and modification (refinement) of the knowledge base without recompilation of the control programs. It is possible in some cases to use different knowledge bases with the same control programs to produce different types of expert systems. Such systems are known as expert system shells since they may be loaded with different knowledge bases.
3. Expert systems are capable of explaining how a particular conclusion was reached, and why requested information is needed during a consultation. This is important as it gives the user a chance to assess and understand the system's reasoning ability, thereby improving the user's confidence in the system.
4. Expert systems use symbolic representations for knowledge (rules, networks or frames) and perform their inference through symbolic computations that closely resemble manipulations of natural language. (An exception to this is an expert system based on neural network architectures.)
5. Expert systems often reason with meta knowledge; that is, they reason with knowledge about themselves, and their own knowledge limits and capabilities.

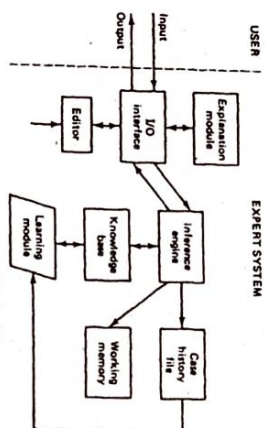


Fig. 1 : Components of a Typical System

The Knowledge Base

The knowledge base contains facts and rules about some specialized knowledge domain. An example of a simple knowledge base giving family relationships is illustrated in Fig. 2. Each fact and rule is identified with a name (a1, a2, ..., r1, r2, ...). For ease in reading, the left side is separated from the right by the implication symbol. Conjunction on the left are given within single parentheses (sublists) and one or more conclusions may follow the implication symbol. Variables are identified as a symbol preceded by a question mark. It should

Artificial Intelligence

be noted that rules found in real working systems may have many conjuncts in the LHS. For example, as many as eight or more are not uncommon.

- ((a1, (male bob))
- (a2, (female sue))
- (a3, (male sam))
- (a4, (male bill))
- (5, (female pam))
- (r1, ((husband ?x ?y))
-
- (male ?x))
- (r2, ((wife ?x ?y))
-
- (female ?x))
- (r3, ((wife ?x ?y))
-
- (husband ?y ?x))
- (r4, ((mother ?x ?y))
-
- (husband ?z ?x))
- (r5, ((father ?x ?y))
-
- (father ?z ?y))
- (r6, ((father ?x ?y))
-
- (wife ?z ?x))
- (r7, ((mother ?x ?y))
-
- (mother ?z ?y))
- (r8, ((husband ?x ?y))
-
- (husband ?z ?y))
- (r9, ((father ?x ?y))
-
- (mother ?z ?y))
- (r10, ((father ?x ?y))
-
- (mother ?z ?y))
- (r11, ((father ?x ?y))
-
- (mother ?z ?y))
- (r12, ((father ?x ?y))
-
- (mother ?z ?y))
- (r13, ((father ?x ?y))
-
- (mother ?z ?y))
- (r14, ((father ?x ?y))
-
- (mother ?z ?y))
- (r15, ((father ?x ?y))
-
- (mother ?z ?y))
- (r16, ((father ?x ?y))
-
- (mother ?z ?y))
- (r17, ((father ?x ?y))
-
- (mother ?z ?y))
- (r18, ((father ?x ?y))
-
- (mother ?z ?y))
- (r19, ((father ?x ?y))
-
- (mother ?z ?y))
- (r20, ((father ?x ?y))
-
- (mother ?z ?y))
- (r21, ((father ?x ?y))
-
- (mother ?z ?y))
- (r22, ((father ?x ?y))
-
- (mother ?z ?y))
- (r23, ((father ?x ?y))
-
- (mother ?z ?y))
- (r24, ((father ?x ?y))
-
- (mother ?z ?y))
- (r25, ((father ?x ?y))
-
- (mother ?z ?y))
- (r26, ((father ?x ?y))
-
- (mother ?z ?y))
- (r27, ((father ?x ?y))
-
- (mother ?z ?y))
- (r28, ((father ?x ?y))
-
- (mother ?z ?y))
- (r29, ((father ?x ?y))
-
- (mother ?z ?y))
- (r30, ((father ?x ?y))
-
- (mother ?z ?y))
- (r31, ((father ?x ?y))
-
- (mother ?z ?y))
- (r32, ((father ?x ?y))
-
- (mother ?z ?y))
- (r33, ((father ?x ?y))
-
- (mother ?z ?y))
- (r34, ((father ?x ?y))
-
- (mother ?z ?y))
- (r35, ((father ?x ?y))
-
- (mother ?z ?y))
- (r36, ((father ?x ?y))
-
- (mother ?z ?y))
- (r37, ((father ?x ?y))
-
- (mother ?z ?y))
- (r38, ((father ?x ?y))
-
- (mother ?z ?y))
- (r39, ((father ?x ?y))
-
- (mother ?z ?y))
- (r40, ((father ?x ?y))
-
- (mother ?z ?y))
- (r41, ((father ?x ?y))
-
- (mother ?z ?y))
- (r42, ((father ?x ?y))
-
- (mother ?z ?y))
- (r43, ((father ?x ?y))
-
- (mother ?z ?y))
- (r44, ((father ?x ?y))
-
- (mother ?z ?y))
- (r45, ((father ?x ?y))
-
- (mother ?z ?y))
- (r46, ((father ?x ?y))
-
- (mother ?z ?y))
- (r47, ((father ?x ?y))
-
- (mother ?z ?y))
- (r48, ((father ?x ?y))
-
- (mother ?z ?y))
- (r49, ((father ?x ?y))
-
- (mother ?z ?y))
- (r50, ((father ?x ?y))
-
- (mother ?z ?y))
- (r51, ((father ?x ?y))
-
- (mother ?z ?y))
- (r52, ((father ?x ?y))
-
- (mother ?z ?y))
- (r53, ((father ?x ?y))
-
- (mother ?z ?y))
- (r54, ((father ?x ?y))
-
- (mother ?z ?y))
- (r55, ((father ?x ?y))
-
- (mother ?z ?y))
- (r56, ((father ?x ?y))
-
- (mother ?z ?y))
- (r57, ((father ?x ?y))
-
- (mother ?z ?y))
- (r58, ((father ?x ?y))
-
- (mother ?z ?y))
- (r59, ((father ?x ?y))
-
- (mother ?z ?y))
- (r60, ((father ?x ?y))
-
- (mother ?z ?y))
- (r61, ((father ?x ?y))
-
- (mother ?z ?y))
- (r62, ((father ?x ?y))
-
- (mother ?z ?y))
- (r63, ((father ?x ?y))
-
- (mother ?z ?y))
- (r64, ((father ?x ?y))
-
- (mother ?z ?y))
- (r65, ((father ?x ?y))
-
- (mother ?z ?y))
- (r66, ((father ?x ?y))
-
- (mother ?z ?y))
- (r67, ((father ?x ?y))
-
- (mother ?z ?y))
- (r68, ((father ?x ?y))
-
- (mother ?z ?y))
- (r69, ((father ?x ?y))
-
- (mother ?z ?y))
- (r70, ((father ?x ?y))
-
- (mother ?z ?y))
- (r71, ((father ?x ?y))
-
- (mother ?z ?y))
- (r72, ((father ?x ?y))
-
- (mother ?z ?y))
- (r73, ((father ?x ?y))
-
- (mother ?z ?y))
- (r74, ((father ?x ?y))
-
- (mother ?z ?y))
- (r75, ((father ?x ?y))
-
- (mother ?z ?y))
- (r76, ((father ?x ?y))
-
- (mother ?z ?y))
- (r77, ((father ?x ?y))
-
- (mother ?z ?y))
- (r78, ((father ?x ?y))
-
- (mother ?z ?y))
- (r79, ((father ?x ?y))
-
- (mother ?z ?y))
- (r80, ((father ?x ?y))
-
- (mother ?z ?y))
- (r81, ((father ?x ?y))
-
- (mother ?z ?y))
- (r82, ((father ?x ?y))
-
- (mother ?z ?y))
- (r83, ((father ?x ?y))
-
- (mother ?z ?y))
- (r84, ((father ?x ?y))
-
- (mother ?z ?y))
- (r85, ((father ?x ?y))
-
- (mother ?z ?y))
- (r86, ((father ?x ?y))
-
- (mother ?z ?y))
- (r87, ((father ?x ?y))
-
- (mother ?z ?y))
- (r88, ((father ?x ?y))
-
- (mother ?z ?y))
- (r89, ((father ?x ?y))
-
- (mother ?z ?y))
- (r90, ((father ?x ?y))
-
- (mother ?z ?y))
- (r91, ((father ?x ?y))
-
- (mother ?z ?y))
- (r92, ((father ?x ?y))
-
- (mother ?z ?y))
- (r93, ((father ?x ?y))
-
- (mother ?z ?y))
- (r94, ((father ?x ?y))
-
- (mother ?z ?y))
- (r95, ((father ?x ?y))
-
- (mother ?z ?y))
- (r96, ((father ?x ?y))
-
- (mother ?z ?y))
- (r97, ((father ?x ?y))
-
- (mother ?z ?y))
- (r98, ((father ?x ?y))
-
- (mother ?z ?y))
- (r99, ((father ?x ?y))
-
- (mother ?z ?y))
- (r100, ((father ?x ?y))
-
- (mother ?z ?y))

Fig. 2 : Facts and rules in a simple knowledge base

The Inference Process

The inference engine accepts user input queries and responses to questions through the I/O interface and uses this dynamic information together with the static knowledge (the rules and facts) stored in the knowledge base. The knowledge in the knowledge base is used to derive conclusions about the current case or situation as presented by the user's input.

The inferring process is carried out recursively in three stages: (1) match, (2) select and (3) execute. During the match stage, the contents of working memory are compared to facts and rules contained in the knowledge base. When consistent matches are found, the corresponding rules are placed in a conflict set. To find an appropriate and consistent match, substitutions (instantiations) may be required. Once all the substituted rules have been added to the conflict set during a given cycle, one of the rules is selected for execution. The criteria for selection may be most recent use, rule condition specificity, (the number of conjuncts on the left) or simply the smallest rule number. The selected rule is then carried out. Fig. 3 illustrates this match select execute cycle.

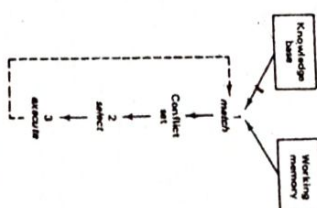


Fig. 3 : The production system inference cycle.

As an example, suppose the working memory contains the two clauses

(father bob sam)
(mother sue sam)

When the match part of the cycle is attempted, a consistent match will be made between these two clauses and rules r_1 and r_2 in the knowledge base. The match is made by substituting bob for ?x, sam for ?z, and sue for ?y. Consequently, since all the conditions on the left of both r_1 and r_2 are satisfied, these two rules will be placed in the conflict set. If there are no other working memory clauses to match, the selection step is executed next. Suppose, for one or more of the selection criteria stated above, r_1 is the rule chosen to execute. The clause on the right side of r_1 is instantiated and the execution step is initiated. The execution step may result in working memory or it may be used to trigger a message to the user. Following the execution step, the match select execute cycle is repeated.

When the left side of a sequence of rules is instantiated first and the rules are executed from left to right, the process is called forward chaining. This is also known as data-driven inference since input data are used to guide the direction of the inference process. For example, we can chain forward to show that when a student is encouraged, is healthy and has goals, the student will succeed.

ENCOURAGED(student) → MOTIVATED(student)
MOTIVATED(student) & HEALTHY(student) → WORKHARD(student) & WORKHARD(student) → EXCELLED(student)
HASGOALS(student) → EXCELLED(student)
EXCELLED(student) → SUCCEEDED(student)

On the other hand, when the right side of the rules is instantiated first, the left-hand conditions become subgoals. These subgoals may in turn cause subgoals to be established, and so on until facts are found to match the lowest subgoal conditions. When this form of inference takes place, we say that backward chaining is performed. This form of inference is also known as goal-driven inference since an initial goal establishes the backward direction of the inferring.

When rules are executed, the resulting action may be the placement of some new facts in working memory, a request for additional information from the user or simply the stopping of the search process. If the appropriate knowledge has been stored in the knowledge base and all required parameter values have been provided by the user, conclusions will be found and will be reported to the user. The chaining continues as long as new matches can be found between clauses in the working memory and rules in the knowledge base. The process stops when no new rules can be placed in the conflict set.

Some systems use both forward and backward chaining, depending on the type of problem and information available.

Many expert systems must deal with uncertain information. This will be the case when the evidence supporting a conclusion is vague, incomplete or otherwise uncertain. To accommodate uncertainties, some form of probabilities, certainty factors, fuzzy logic, heuristics or other methods must be introduced into the inference process.

Explaining How or Why

The explanation module provides the user with an explanation of the reasoning process when requested. This is done in response to a how query or a why query. To respond to a how query, the explanation module traces the chain of rules fired during a consultation with the user. The sequence of rules that led to the conclusion is then printed for the user in an easy-to-understand human-language style. This permits the user to actually see the reasoning process followed by

the system in arriving at the conclusion. If the user does not agree with the reasoning steps presented, they may be changed using the editor.

To respond to a why query, the explanation module must be able to explain why certain information is needed by the inference engine to complete a step in the reasoning process before it can proceed. For example, in diagnosing a car that will not start, a system might be asked why it needs to know the status of the distributor spark. In response, the system would reply that it needs this information to determine if the problem can be isolated to the ignition system. Again, this information allows the user to determine if the system's reasoning steps appear to be sound. The explanation module programs give the user the important ability to follow the inferring steps at any time during the consultation.

Building a Knowledge Base

One of the most difficult tasks in creating and maintaining production systems is the building and maintaining of consistent but complete set of rules. This should be done without adding redundant or unnecessary rules. Building knowledge base requires careful planning, accounting an organization of the knowledge structures. It also requires thorough validation and verification of the complete knowledge base, operations which have yet to be perfected. An "intelligent" editor can greatly simplify the process of building a knowledge base.

The I/O Interface

The input-output interface permits the user to communicate with the system in a more natural way by permitting the use of simple selection means or the use of restricted language which is close to a natural language. This means that the system must have special prompts or specialized vocabulary which encompasses the terminology of the given domain of expertise. For example, MYCIN can recognize many medical terms in addition to various common words needed to communicate. For this, MYCIN has vocabulary of some 2000 words.

The learning module and history file are not components of expert systems. When they are provided the are used to assist in building and refining the knowledge base.

Examples of Expert Systems :

(i) **Dendral**: Dendral was an influential pioneer project in artificial intelligence (AI) of the 1960s and the computer software expert system that it produced. Its primary aim was to study hypothesis formation and discovery in science. For that, a specific task in science was chosen, help organic chemists in identifying unknown organic molecules, by analyzing their mass spectra and using knowledge of chemistry. It was done at Stanford University by Edwin Feigenbaum, Bruce Buchanan, Joshua Lederberg and Ca

Djerassi, along with a team of highly creative research associates and students. It began in 1965 and spans approximately half the history of AI research.

The software program Dendral is considered the first expert system because it automated the decision-making process and problem-solving behaviour of organic chemists. The project consisted of research on two main programs: Heuristic Dendral and Meta-Dendral and several sub-programs. It was written in Lisp (programming language), which was considered the language of AI because of its flexibility.

Many systems were derived from Dendral, including MYCIN, MOLGEN, MACSYMA, PROSPECTOR, XCON and STEAMER. The name Dendral is a portmanteau of the term "Dendritic Algorithm".

(ii) **MYCIN**: MYCIN was an early expert system that used artificial intelligence to identify bacteria causing severe infections, such as bacteremia and meningitis and to recommend antibiotics, with the dosage adjusted for patient's body weight, the name derived from the suffix "mycin". The Mycin as many antibiotics have the suffix "mycin". The Mycin system was also used for the diagnosis of blood clotting diseases.

MYCIN was developed over five or six years in the early 1970s at Stanford University. It was written in Lisp as the doctoral dissertation of Edward Shortliffe under the direction of Bruce Buchanan, Stanley N. Cohen and others. It arose in the laboratory that had created the earliest Dendral expert system. MYCIN was never actually used in practice but research indicated that it proposed an acceptable therapy in about 69% of cases, which was better than the performance of infectious disease experts who were judged using the same criteria.

Expert System Features : There are a number of features which are commonly used in expert systems. These features allow the users to fully utilize the expert system's capability conveniently in providing the most logical and reasonable decision in a problematic situation.

- **Backward chaining** – an inference technique which continuously break a goal into smaller sub-goals which are easier to prove via IF THEN rules
- **Dealing with uncertainties** – the system has the capability to handle and reason with conditions that are uncertain and data which are not precisely known
- **Forward chaining** – an inference technique which deduce a problem solution from initial data via IF THEN rules
- **Data representation** – the method where the specific problem data is stored and accessed in the system

- User interface – that portion of the code which creates an easy to use system;
- **Explanations** – the ability of the system to explain the reasoning process that it used to reach a recommendation.

Prob.13 Discuss about the progressive advancement in robots by giving suitable examples. [R.T.U. 2017]

OR

What is progressive advancement in robotics? Explain. [R.T.U. 2018]

OR

Discuss about the Progressive Advancement in Robots by giving suitable examples. [R.T.U. Dec. 2013]

OR

What is progressive advancement in Robotics? [R.T.U. 2014]

Sol. Progressive Advancement in Robots : The growth in the capabilities of robots has been taking rapid strides since the introduction of robots in the industry in early 1960s, but there is still a long way to go to obtain the super-humanoid anthropomorphic robot depicted in fiction. The growth of robots can be ground into robot generations, based on characteristics breakthroughs in robot's capabilities. These generations are overlapping and include futuristic projections.

1. **First Generation**: The first generation robots are repeating, nonservo, pick-and-place, or point-to-point kind. The technology for these is fully developed and at present about 80% robots in use in the industry are of this kind. It is pre-tied that these will continue to be in use for a long time.

2. **Second Generation**: The addition of sensing devices and enabling the robot to alter its movements in response to sensory feedback marked the beginning of second generation. These robots exhibit path-control capabilities. This technological breakthrough came around 1980s and is yet not mature.

3. **Third Generation**: The third generation is marked with robots having human-like intelligence. The growth in computers led to high-speed processing of information and thus, robots also acquired artificial intelligence, self-learning, and conclusion-drawing capabilities by past experiences. On-line computations and control, artificial vision, and active force/torque interaction with the environment are the significant characteristics of these robots. The technology is still in infancy and has to go a long way.

4. **Fourth Generation**: This is futuristic and may be a reality only during this millennium. Prediction about its features is difficult, is not impossible. It may be a true android or an artificial biological robot or a super humanoid capable

of producing its own clones. This might provide for fifth and higher generation robots.

A pictorial visualization of these overlapping generations of robots is given in Fig.

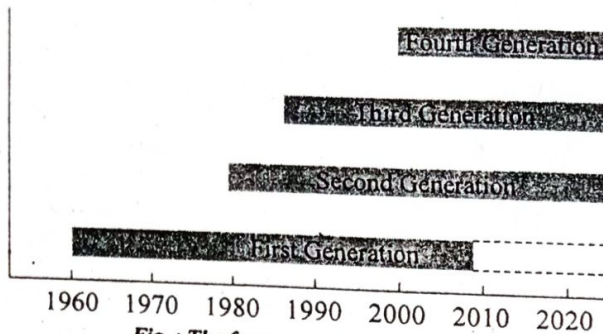


Fig. : The four generations of robots

Examples of Progressive Advancement : Well it is a system that contains sensors, control systems, manipulators, power supplies and software all working together to perform a task. Designing, building, programming and testing a robot is a combination of physics, mechanical engineering, electrical engineering, structural engineering, mathematics and computing. In some cases biology, medicine, chemistry might also be involved. A study of robotics means that students are actively engaged with all of these disciplines in a deeply problem-posing problem-solving environment.

The mature advancement of technology is required for the development of the robotics in the world emerging technology. Emerging technologies include a variety of technologies such as educational technology, information technology, nanotechnology, biotechnology, cognitive science, psycho technology, robotics, and artificial intelligence etc.

(i) **Sensing :** The 3-D measurements are useful for autonomous mobile robot to recognize its environments as well as the reconstruction of 3-D information of large scale structures in construction sites. The combination of a LRF (Laser Range Finder) and a camera can reconstruct the inside of a building. Images from the camera are mapped on the corresponding 3-D scan data from LRF by using texture mapping technique. The SLAM is useful technique to achieve map building and robot localization at the same time.

(ii) **Actuation :** The dexterous hand mechanisms are capable of manipulating and handling various objects in construction sites. A unique finger mechanism is proposed with omni directional driving roller to realize the two active rotational axes on the surface of the grasped object. The cylindrical tracked unit can be applied as roller and the fingers can manipulate the grasped object in the arbitrary axes.

(iii) **Mobile Robot :** Mobile manipulation is one of the active research areas. The omni directional mobile capability is useful in complicated construction environments. The mobility can be more enhanced by introducing an active caster mechanism. The active-caster wheel features simple mechanism and high

positioning accuracy, and will allow a robot to carry out more accurate object transportation. Development of leg robot is still active and expected in the rough terrain tasks as well as in narrow and small spaces in the inside of buildings and large structures for inspection and monitoring tasks.

(iv) **Power Assist :** The power assist system has its long history in its development starting in early 1960s at General Electric. The recent advancement of the computer technology has brought a chance to realization of practical power assist mechanism. The main objective is to support weakened senior people and the disabled in their daily life as well as to assist workers in heavy duties which are common in construction sites.

Prob.14 Describe the industrial applications of robots.

[R.T.U. 2016]

OR

Justify application of robots in industries and enumerates its advantages.

[R.T.U. 2015]

OR

Write short note on industrial application of Robots.

[R.T.U. Dec. 2013]

OR

What are the different categories of robot industrial applications?

[R.T.U. 2013]

Sol. Application of Robots In Industries : Robots are employed in a wide assortment of application in industry. Today most of the applications are in manufacturing to move materials, parts, and tools of various types. Future application will include nonmanufacturing tasks, such as construction work, exploration of space and medical care. At some time in the distant future, a household robot may become a mass produced item, perhaps as common place as the automobile is today.

For the present, most industrial applications of robots can be divided into the following three categories :

1. **Material-Handling and Machine-Loading and Unloading Applications :** In these applications the robot's function is to move materials or parts from one location in the work cell to some other location. The MAKER 110 in Fig. 2 is shown performing a material handling operation. Loading and/or unloading of production machine is included within the scope of the material-handling activity. The Unimate 2000 in fig. is performing a machine load/unload operation for machine tool.