UNIT – 1 (QUESTION BANK)

1	Explain AI problems and techniques in detail.
2	Explain AI Programming Languages.
3	Explain Blind search strategy with BFS and DFS.
4	Explain Breath First and Depth First search.
5	Explain Heuristic search algorithm.
6	Explain Hill climbing algorithm.
7	Explain Game playing algorithm.
8	Explain Alpha beta pruning.
9	What is AI technique? Explain in detail.
10	Define production system. What are the various types of production system?
11	Write AO* algorithm and explain with an example.
12	What are the problems in hill climbing search methods due to which they may fail to find the solutions?
13	Explain MIN-MAX algorithm.
14	Explain foundation and history of AI.
15	Explain A* algorithm in detail.

UNIT – 1 (NOTES)

What is Artificial Intelligence?

According to the father of Artificial Intelligence, John McCarthy, it is "The science and engineering of making intelligent machines, especially intelligent computer programs". Artificial Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner the intelligent humans think. AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

Philosophy of AI

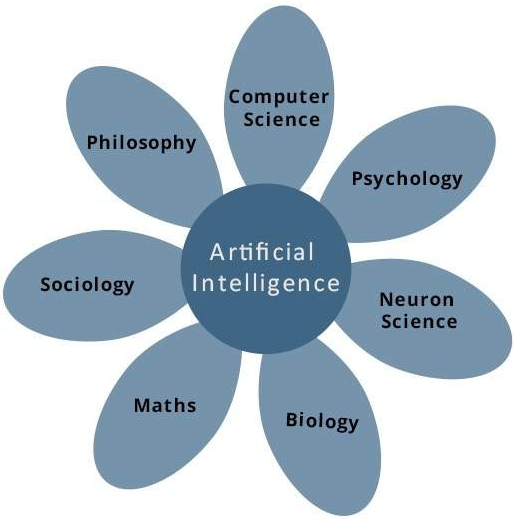
While exploiting the power of the computer systems, the curiosity of human, lead him to wonder, "Can a machine think and behave like humans do?" Thus, the development of AI started with the intention of creating similar intelligence in machines that we find and regard high in humans.

Goals of AI

- **To Create Expert Systems** – The systems which exhibit intelligent behavior, learn, demonstrate, explain, and advice its users.
- **To Implement Human Intelligence in Machines** – Creating systems that understand, think, learn, and behave like humans.

What Contributes to AI?

Artificial intelligence is a science and technology based on disciplines such as Computer Science, Biology, Psychology, Linguistics, Mathematics, and Engineering. A major thrust of AI is in the development of computer functions associated with human intelligence, such as reasoning, learning, and problem solving. Out of the following areas, one or multiple areas can contribute to build an intelligent system.



Programming Without and With AI

The programming without and with AI is different in following ways –

Programming Without AI	Programming With AI
A	A

computer program without AI can answer the **specific** questions it is meant to solve.	computer program with AI can answer the **generic** questions it is meant to solve.
Modification in the program leads to change in its structure.	AI programs can absorb new modifications by putting highly independent pieces of information together. Hence you can modify even a

	minute piece of information of program without affecting its structure.
Modification is not quick and easy. It may lead to affecting the program adversely.	Quick and Easy program modification.

What is AI Technique?

In the real world, the knowledge has some unwelcomed properties –

- Its volume is huge, next to unimaginable.
- It is not well-organized or well-formatted.
- It keeps changing constantly.

AI Technique is a manner to organize and use the knowledge efficiently in such a way that –

- It should be perceivable by the people who provide it.
- It should be easily modifiable to correct errors.
- It should be useful in many situations though it is incomplete or inaccurate.

AI techniques elevate the speed of execution of the complex program it is equipped with.

Applications of AI

AI has been dominant in various fields such as –

- **Gaming** – AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.
- **Natural Language Processing** – It is possible to interact with the computer that understands natural language spoken by humans.
- **Expert Systems** – There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
- **Vision Systems** – These systems understand, interpret, and comprehend visual input on the computer. For example,
    - A spying aeroplane takes photographs, which are used to figure out spatial information or map of the areas.
    - Doctors use clinical expert system to diagnose the patient.
    - Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.
- **Speech Recognition** – Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.
- **Handwriting Recognition** – The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.
- **Intelligent Robots** – Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition,

they are capable of learning from their mistakes and they can adapt to the new environment.

History of AI

Here is the history of AI during 20th century –

Year	Milestone / Innovation
1923	Karel Čapek play named “Rossum's Universal Robots” (RUR) opens in London, first use of the word "robot" in English.
1943	Foundations for neural networks laid.
1945	Isaac Asimov, a Columbia University alumni, coined the term Robotics.
1950	Alan Turing introduced Turing Test for evaluation of intelligence and published Computing Machinery and Intelligence. Claude Shannon published Detailed Analysis of Chess Playing as a search.
1956	John McCarthy coined the term Artificial Intelligence. Demonstration of the first running AI program at Carnegie Mellon University.
1958	John McCarthy invents LISP programming language for AI.
1964	Danny Bobrow's dissertation at MIT showed that computers can understand natural language well enough to solve algebra word problems correctly.
1965	Joseph Weizenbaum at MIT built ELIZA, an interactive problem that carries on a dialogue in English.
1969	Scientists at Stanford Research Institute Developed Shakey, a robot, equipped with locomotion, perception, and problem solving.
1973	The Assembly Robotics group at Edinburgh University built Freddy, the Famous Scottish Robot, capable of using vision to locate and assemble models.
1979	The first computer-controlled autonomous vehicle, Stanford Cart, was built.

1985	Harold Cohen created and demonstrated the drawing program, Aaron.
1990	Major advances in all areas of AI – • Significant demonstrations in machine learning • Case-based reasoning • Multi-agent planning • Scheduling • Data mining, Web Crawler • natural language understanding and translation • Vision, Virtual Reality • Games
1997	The Deep Blue Chess Program beats the then world chess champion, Garry Kasparov.
2000	Interactive robot pets become commercially available. MIT displays Kismet, a robot with a face that expresses emotions. The robot Nomad explores remote regions of Antarctica and locates meteorites.

LISP

It serves as a common language, which can be easily extended for specific implementation. Programs written in Common LISP do not depend on machine-specific characteristics, such as word length etc.

Features of Common LISP

- It is machine-independent
- It uses iterative design methodology, and easy extensibility.
- It allows updating the programs dynamically.
- It provides high level debugging.
- It provides advanced object-oriented programming.
- It provides a convenient macro system.

- It provides wide-ranging data types like, objects, structures, lists, vectors, adjustable arrays, hash-tables, and symbols.
- It is expression-based.
- It provides an object-oriented condition system.
- It provides a complete I/O library.
- It provides extensive control structures.

Applications Built in LISP

Large successful applications built in Lisp.

- Emacs
- G2
- AutoCad
- Igor Engraver
- Yahoo Store

What is Prolog?

- Prolog stands for **Programming in logic.** It is used in artificial intelligence programming.
- Prolog is a **declarative** programming language.

For example: While implementing the solution for a given problem, instead of specifying the ways to achieve a certain goal in a specific situation, user needs to specify about the situation (rules and facts) and the goal (query). After these stages, Prolog interpreter derives the solution.

- Prolog is useful in AI, NLP, databases but useless in other areas such as graphics or numerical algorithms.

Prolog facts

- A fact is something that seems to be true.

For example: It's raining.

- In Prolog, facts are used to form the statements. Facts consist of a specific item or relation between two or more items.

How to convert English to prolog facts using facts and rules?

It is very simple to convert English sentence into Prolog facts. Some examples are explained in the following table.

English Statements	Prolog Facts
Dog is barking	barking(dog)
Jaya likes food if it is delicious.	likes(Jaya, Food):-delicious(Food)

In the above table, the statement **'Dog is barking'** is a **fact,** while the statement **'Jaya likes food if it is delicious'** is called **rule.** In this statement, variable like 'Food' has a first letter in capital, because its value came from previous fact. The symbol **':-'** is used to denote that "Jaya likes delicious food".

Advantages:

1. Easy to build database. Doesn't need a lot of programming effort.

2. Pattern matching is easy. Search is recursion based.

3. It has built in list handling. Makes it easier to play with any algorithm involving lists.

Disadvantages:

1. LISP (another logic programming language) dominates over prolog with respect to I/O features.

2. Sometimes input and output is not easy.

Applications:

Prolog is highly used in artificial intelligence (AI). Prolog is also used for pattern matching over natural language parse trees.

Uninformed/Blind Search:

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search.It examines each node of the tree until it achieves the goal node.

It can be divided into five main types:

- Breadth-first search
- Uniform cost search
- Depth-first search
- Iterative deepening depth-first search
- Bidirectional Search

Informed Search

Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a

Heuristic search.

A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time. Informed search can solve much complex problem which could not be solved in another way. An example of informed search algorithms is a traveling salesman problem.

1. Greedy Search
2. A* Search

Uninformed Search Algorithms

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

1. Breadth-first Search
2. Depth-first Search
3. Depth-limited Search
4. Iterative deepening depth-first search
5. Uniform cost search
6. Bidirectional Search

1. Breadth-first Search:

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

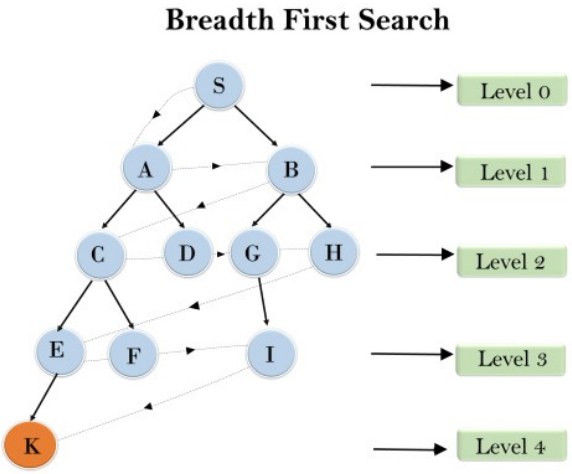
Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

1. S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K



Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

$T(b) = 1+b^2+b^3+.....+ b^d = O(b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

2. Depth-first Search

- Depth-first search isa recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.

- The process of the DFS algorithm is similar to the BFS algorithm

Advantages:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

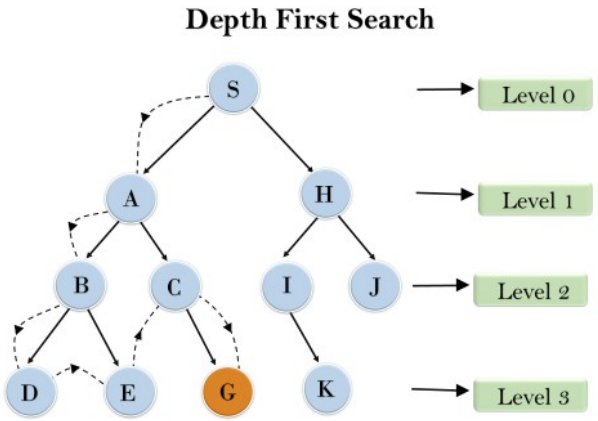
- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node ----> right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$T(n)= 1+ n^2+ n^3 +.....+ n^m=O(n^m)$

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node

Informed Search Algorithms

So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristics function: Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

1. $h(n) <= h^*(n)$

Here h(n) is heuristic cost, and $h^*(n)$ is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

Pure Heuristic Search:

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.

In the informed search we will discuss two main algorithms which are given below:

Best First Search Algorithm (Greedy search)

- **A* Search Algorithm**

1.) Best-first Search Algorithm (Greedy Search):

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

1. f(n)= g(n).

Were, h(n)= estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the priority queue.

Best first search algorithm:

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- **Step 4:** Expand the node n, and generate the successors of node n.

- **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

Advantages:

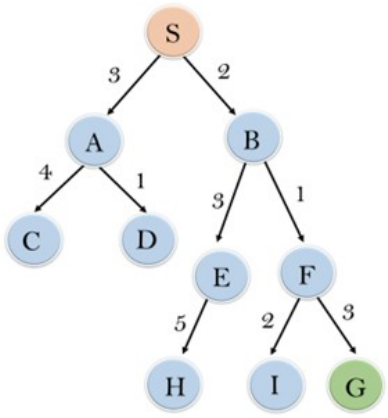
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

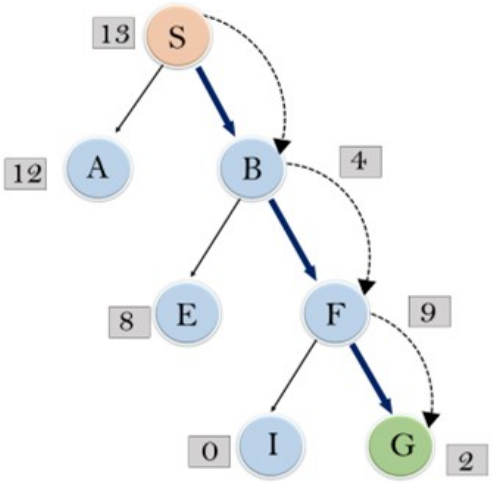
Example:

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

In this search example, we are using two lists which are OPEN and CLOSED Lists. Following are the iteration for traversing the above example.



Expand the nodes of S and put in the CLOSED list

Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B]

: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]

: Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be: **S----> B----->F----> G**

Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.

Space Complexity: The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

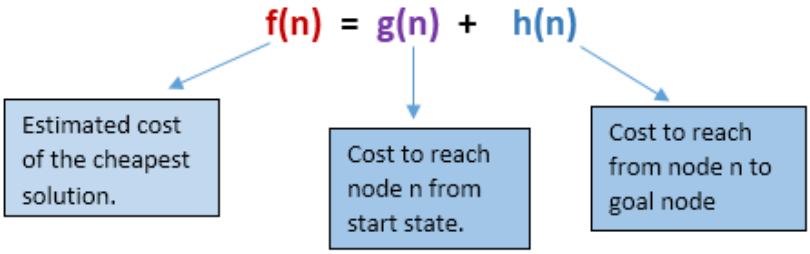
Optimal: Greedy best first search algorithm is not optimal.

2.) A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS

and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



At each point in the search space, only those node is expanded which have the lowest value of f(n), and the algorithm terminates when the goal node is found.

Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.

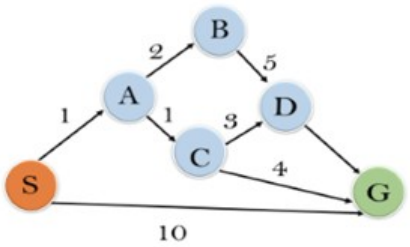
- This algorithm can solve very complex problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

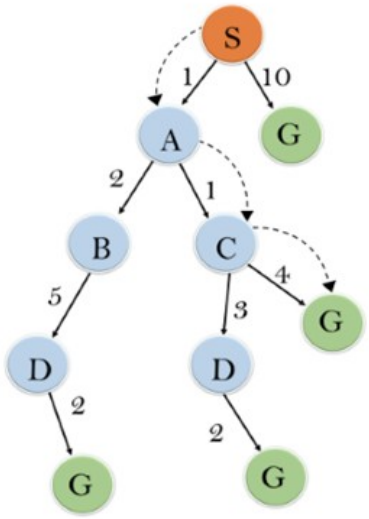
Example:

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.
Here we will use OPEN and CLOSED list.



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution:



Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

Hill Climbing Algorithm in Artificial Intelligence

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

Features of Hill Climbing:

Following are some main features of Hill Climbing Algorithm:

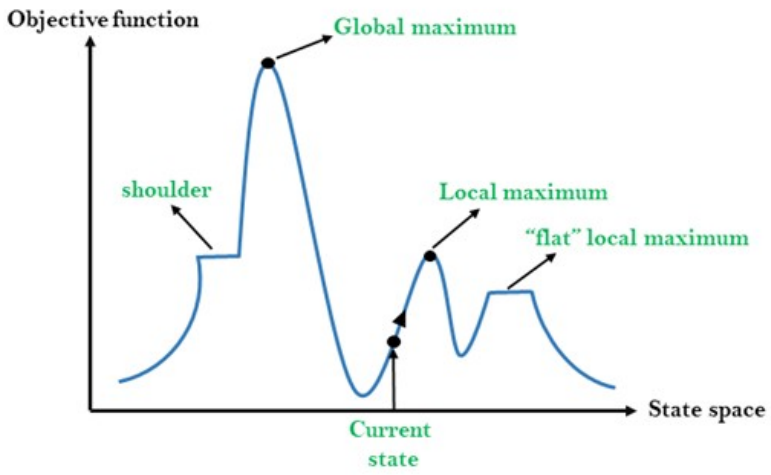
- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.

- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing:

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.

On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.



Different regions in the state space landscape:

Local Maximum: Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.

Global Maximum: Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

Current state: It is a state in a landscape diagram where an agent is currently present.

Flat local maximum: It is a flat space in the landscape where all the neighbor states of current states have the same value.

Shoulder: It is a plateau region which has an uphill edge.

Types of Hill Climbing Algorithm:

- Simple hill Climbing:
- Steepest-Ascent hill-climbing:
- Stochastic hill Climbing:

1. Simple Hill Climbing:

Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:

- Less time consuming
- Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
    1. If it is goal state, then return success and quit.
    2. Else if it is better than the current state then assign new state as a current state.
    3. Else if not better than the current state, then return to step2.

b. **Step 5:** Exit.

2. Steepest-Ascent hill climbing:

The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
    1. Let SUCC be a state such that any successor of the current state will be better than it.
    2. For each operator that applies to the current state:
        I. Apply the new operator and generate a new state.
        II. Evaluate the new state.
        III. If it is goal state, then return it and quit, else compare it to the SUCC.
        IV. If it is better than SUCC, then set new state as SUCC.
        V. If the SUCC is better than the current state, then set current state to SUCC.

b. **Step 5:** Exit.

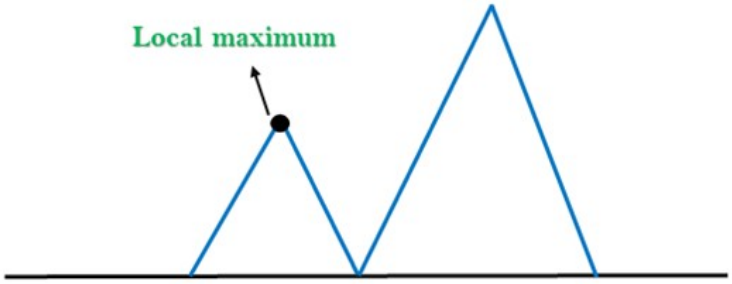
3. Stochastic hill climbing:

Stochastic hill climbing does not examine for all its neighbor before moving. Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

Problems in Hill Climbing Algorithm:

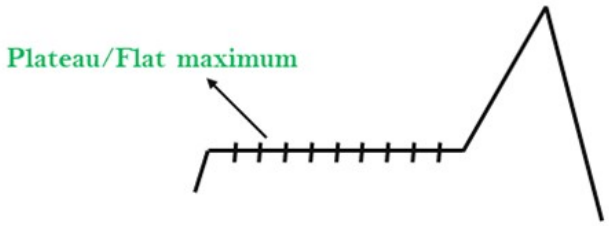
1. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

Solution: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.



3. Ridges: A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.

Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.

Simulated Annealing:

A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum. And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient. **Simulated Annealing** is an algorithm which yields both efficiency and completeness. In mechanical

term **Annealing** is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state. The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move. If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.

A* Algorithm

- A* Algorithm is one of the best and popular techniques used for path finding and graph traversals.
- A lot of games and web-based maps use this algorithm for finding the shortest path efficiently.
- It is essentially a best first search algorithm.

A* Algorithm works as-

- It maintains a tree of paths originating at the start node.
- It extends those paths one edge at a time.
- It continues until its termination criterion is satisfied.

A* Algorithm extends the path that minimizes the following function-

Here,

- 'n' is the last node on the path
- g(n) is the cost of the path from start node to node 'n'
- h(n) is a heuristic function that estimates cost of the cheapest path from node 'n' to the goal node.

Algorithm Steps:

- The implementation of A* Algorithm involves maintaining two lists- OPEN and CLOSED.

- OPEN contains those nodes that have been evaluated by the heuristic function but have not been expanded into successors yet.
- CLOSED contains those nodes that have already been visited.

The algorithm is as follows-

Step-01:

- Define a list OPEN.
- Initially, OPEN consists solely of a single node, the start node S.

Step-02:

If the list is empty, return failure and exit.

Step-03:

- Remove node n with the smallest value of f(n) from OPEN and move it to list CLOSED.
- If node n is a goal state, return success and exit.

Step-04:

Expand node n.

Step-05:

- If any successor to n is the goal node, return success and the solution by tracing the path from goal node to S.
- Otherwise, go to Step-06.

Step-06:

For each successor node,

- Apply the evaluation function f to the node.
- If the node has not been in either list, add it to OPEN.

Step-07:

Go back to Step-02.

AO* Search: (And-Or) Graph

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

Like A* algorithm here we will use two arrays and one heuristic function.

OPEN:

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

CLOSE:

It contains the nodes that have already been processed.

6 7: The distance from current node to goal node.

Algorithm:

Step 1: Place the starting node into OPEN.

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

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

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

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

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

Step 7: Return to Step 2.

Step 8: Exit.

Mini-Max Algorithm in Artificial Intelligence:

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Pseudo-code for MinMax Algorithm:

```
1.  function minimax(node, depth, maximizingPlayer) is
2.  if depth ==0 or node is a terminal node then
3.  return static evaluation of node
4.
5.  if MaximizingPlayer then      // for Maximizer Player
6.  maxEva= -infinity
7.   for each child of node do
8.   eva= minimax(child, depth-1, false)
```

```
9.          maxEva= max(maxEva,eva)       //gives Maximum of the values
10.         return maxEva
11.
12.         else                  // for Minimizer player
13.          minEva= +infinity
14.          for each child of node do
15.          eva= minimax(child, depth-1, true)
16.          minEva= min(minEva, eva)       //gives minimum of the values
17.          return minEva
```

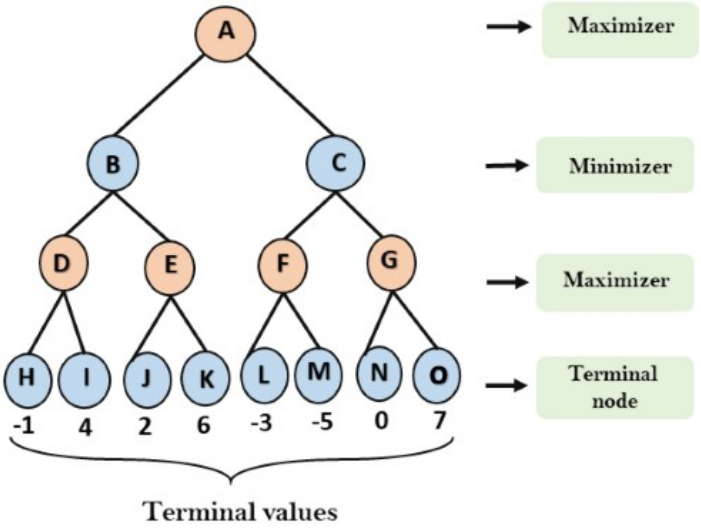
Initial call:

Minimax(node, 3, true)

Working of Min-Max Algorithm:

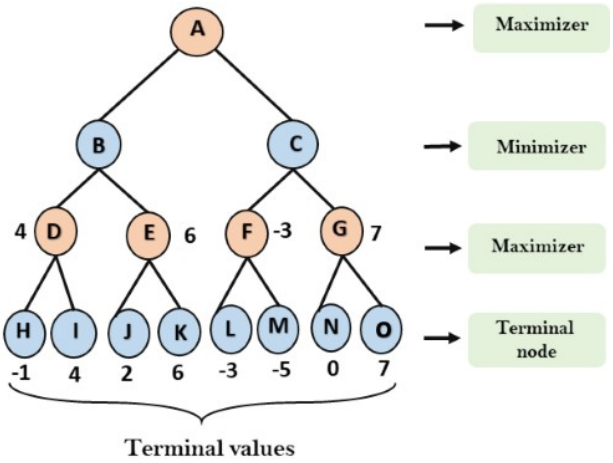
- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



Step 2: Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

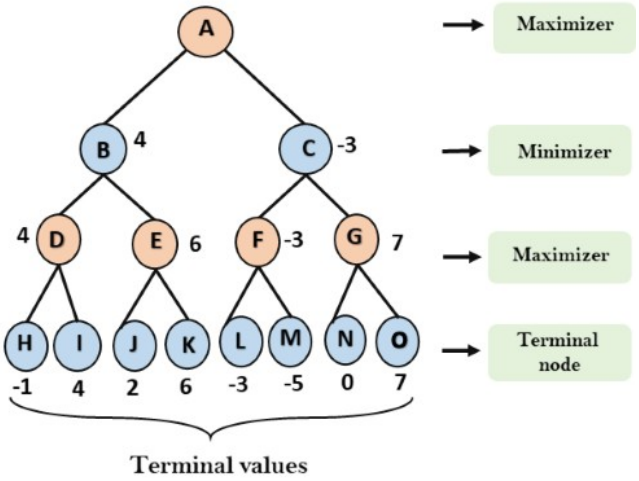
- For node D max(-1,- -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

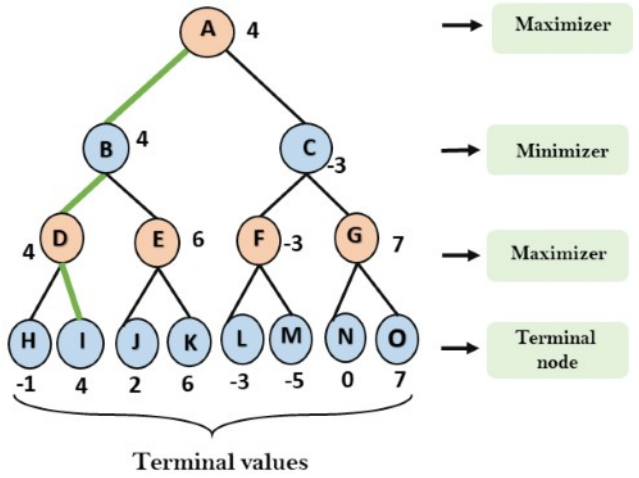
- For node B= min(4,6) = 4

- For node C= min (-3, 7) = -3



Step 3: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A max(4, -3)= 4



That was the complete workflow of the minimax two player game.

Properties of Mini-Max algorithm:

- **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.

- **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is **$O(b^m)$**, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is **O(bm)**.

Limitation of the minimax Algorithm:

The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of games has a huge branching factor, and the player has lots of choices to decide. This limitation of the minimax algorithm can be improved from **alpha-beta pruning** which we have discussed in the next topic.

Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
    1. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
    2. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.

b. The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Note: To better understand this topic, kindly study the minimax algorithm.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

1. α>=β

Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Pseudo-code for Alpha-beta Pruning:

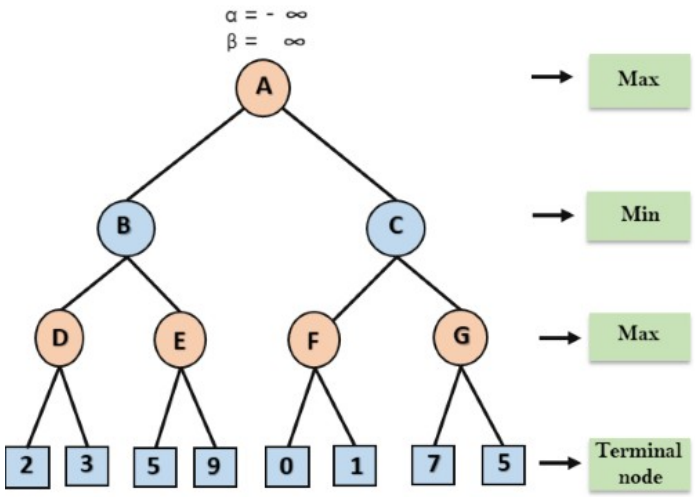
```
1.  function minimax(node, depth, alpha, beta, maximizingPlayer) is
2.  if depth ==0 or node is a terminal node then
3.  return static evaluation of node
4.
5.  if MaximizingPlayer then      // for Maximizer Player
6.    maxEva= -infinity
7.    for each child of node do
8.    eva= minimax(child, depth-1, alpha, beta, False)
9.   maxEva= max(maxEva, eva)
10.  alpha= max(alpha, maxEva)
11.   if beta<=alpha
12.  break
13.  return maxEva
14.
```

```
15.        else                    // for Minimizer player
16.          minEva= +infinity
17.          for each child of node do
18.          eva= minimax(child, depth-1, alpha, beta, true)
19.          minEva= min(minEva, eva)
20.          beta= min(beta, eva)
21.           if beta<=alpha
22.          break
23.          return minEva
```

Working of Alpha-Beta Pruning:

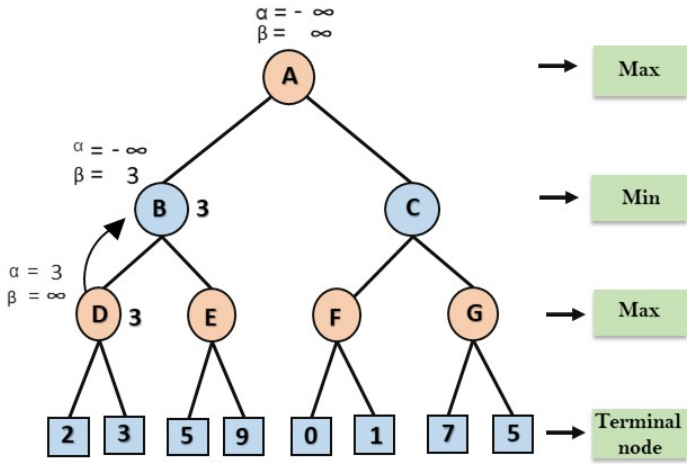
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



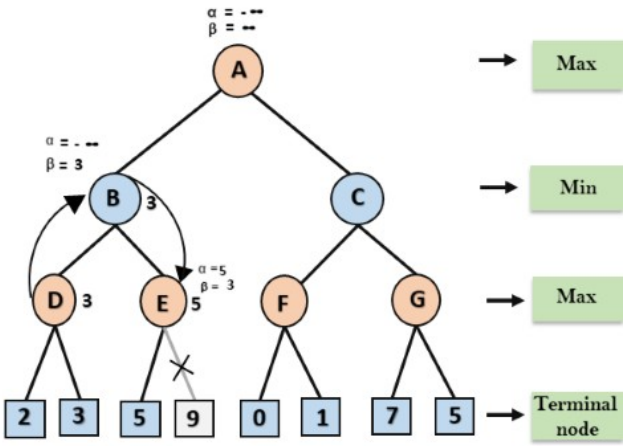
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

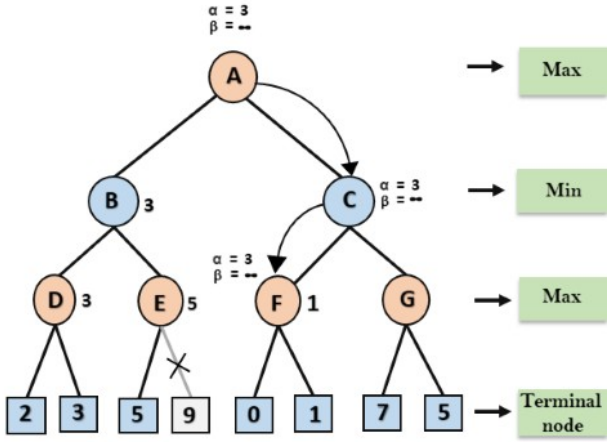
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



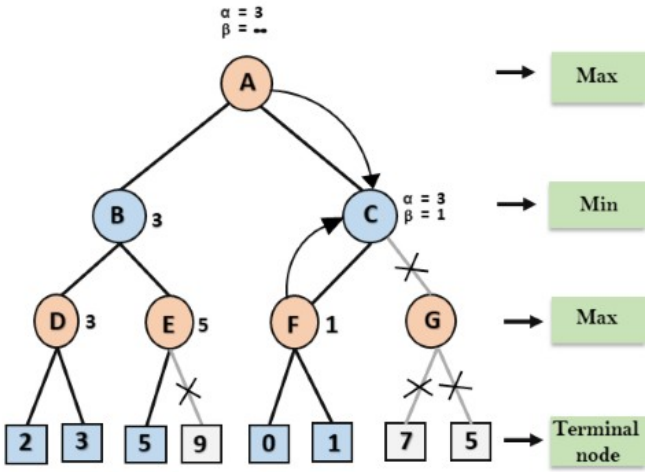
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

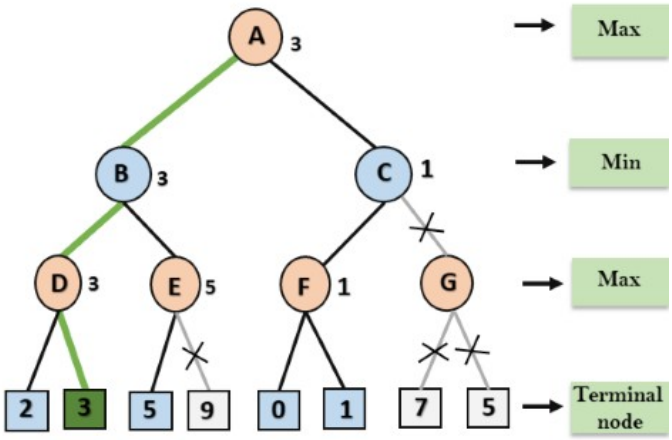
Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Move Ordering in Alpha-Beta pruning:

The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.

It can be of two types:

- **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
- **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

Rules to find good ordering:

Following are some rules to find good ordering in alpha-beta pruning:

- Occur the best move from the shallowest node.
- Order the nodes in the tree such that the best nodes are checked first.
- Use domain knowledge while finding the best move. Ex: for Chess, try order: captures first, then threats, then forward moves, backward moves.
- We can bookkeep the states, as there is a possibility that states may repeat.

UNIT – 2 (QUESTION BANK)

16	Explain knowledge representation and its issues.
17	Explain prediction logic and logic programming.
18	Explain semantic nets, frames, and inheritance.
19	How is knowledge representation in semantic nets?
20	Draw a semantic network to represent: Every teacher like intelligent students.
21	What is frame? What are the reasoning actions that can be performed using frames?
22	Explain constraint propagation.
23	Explain representing knowledge using rules.
24	Explain rules based deduction system.

UNIT – 2 (NOTES)

What is knowledge representation?

Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. But how machines do all these things comes under knowledge representation and reasoning. Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent:

Following are the kind of knowledge which needs to be represented in AI systems:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

Types of knowledge

Following are the various types of knowledge:

1. Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarative sentences.

- It is simpler than procedural language.

2. Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

4. Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a filed or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

5. Structural knowledge:

- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

Issues in knowledge representation

The main objective of knowledge representation is to draw the conclusions from the knowledge, but there are many issues associated with the use of knowledge representation techniques.

Some of them are listed below:

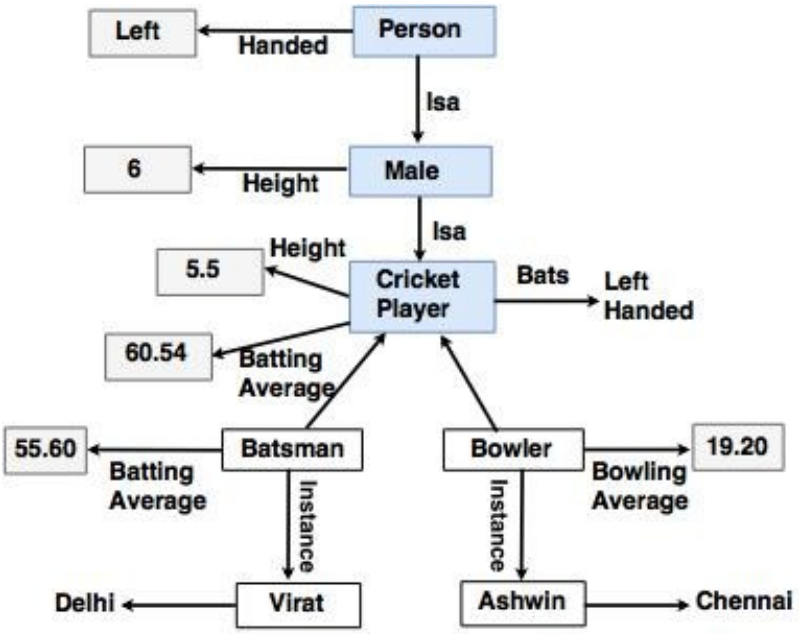


Fig: Inheratable Knowledge Representation

Refer to the above diagram to refer to the following issues.

1. Important attributes

There are two attributes shown in the diagram, **instance** and **isa.** Since these attributes support property of inheritance, they are of prime importance.

2. Relationships among attributes

Basically, the attributes used to describe objects are nothing but the entities. However, the attributes of an object do not depend on the encoded specific knowledge.

3. Choosing the granularity of representation

High-level facts may be insufficient to draw the conclusion while Low-level primitives may require a lot of storage.

For example: Suppose that we are interested in following facts:

John spotted Alex.

Now, this could be represented as "Spotted (agent(John), object (Alex))"

Such a representation can make it easy to answer questions such as: Who spotted Alex?

Suppose we want to know: "Did John see Sue?"

Given only one fact, user cannot discover that answer.

Hence, the user can add other facts, such as "Spotted (x, y) → saw (x, y)"

4. Representing sets of objects.

There are some properties of objects which satisfy the condition of a set together but not as individual;

Example: Consider the assertion made in the sentences:

"There are more sheep than people in Australia", and "English speakers can be found all over the world."

These facts can be described by including an assertion to the sets representing people, sheep, and English.

5. Finding the right structure as needed

To describe a particular situation, it is always important to find the access of right structure. This can be done by selecting an initial structure and then revising the choice. While selecting and reversing the right structure, it is necessary to solve following problem statements. They include the process on how to:

Select an initial appropriate structure.

- Fill the necessary details from the current situations.
- Determine a better structure if the initially selected structure is not appropriate to fulfill other conditions.
- Find the solution if none of the available structures is appropriate.
- Create and remember a new structure for the given condition.
- There is no specific way to solve these problems, but some of the effective knowledge representation techniques have the potential to solve them.

Predicate

A predicate is an expression of one or more variables defined on some specific domain. A predicate with variables can be made a proposition by either assigning a value to the variable or by quantifying the variable. Consider the following statement.

- Ram is a student.

Now consider the above statement in terms of Predicate calculus.

- Here "is a student" is a predicate and Ram is subject.
- Let's denote "Ram" as x and "is a student" as a predicate P then we can write the above statement as P(x).
- Generally a statement expressed by Predicate must have at least one object associated with Predicate. In our case, Ram is the required object with associated with predicate P.

Statement Function

Earlier we denoted "Ram" as x and "is a student" as predicate P then we have statement as P(x). Here P(x) is a statement function where if we replace x with a Subject say Sunil then we'll be having a statement "Sunil is a student." Thus a statement function is an expression having Predicate Symbol and one or multiple variables. This statement function gives a statement when we replaced the variables with objects. This replacement is called substitution instance of statement function.

Quantifiers

The variable of predicates is quantified by quantifiers. There are two types of quantifier in predicate logic – Universal Quantifier and Existential Quantifier.

Universal Quantifier

Universal quantifier states that the statements within its scope are true for every value of the specific variable. It is denoted by the symbol ∀.

∀ x P(x) is read as for every value of x, P(x) is true.

Example – "Man is mortal" can be transformed into the propositional form ∀ x P(x) where P(x) is the predicate which denotes x is mortal and ∀ x represents all men.

Existential Quantifier

Existential quantifier states that the statements within its scope are true for some values of the specific variable. It is denoted by the symbol ∃.

∃ x P(x) is read as for some values of x, P(x) is true.

Example – "Some people are dishonest" can be transformed into the propositional form ∃ x P(x) where P(x) is the predicate which denotes x is dishonest and ∃ x represents some dishonest men.

Predicate Formulas

Consider a Predicate P with n variables as $P(x_1, x_2, x_3, ..., x_n)$. Here P is n-place predicate and $x_1, x_2, x_3, ..., x_n$ are n individuals variables. This n-place predicate is known as atomic formula of predicate calculus. For Example: P(), Q(x, y), R(x,y,z)

Well Formed Formula

Well Formed Formula (wff) is a predicate holding any of the following –

- All propositional constants and propositional variables are wffs
- If x is a variable and Y is a wff, ∀ x Y and ∀ x Y are also wff
- Truth value and false values are wffs
- Each atomic formula is a wff
- All connectives connecting wffs are wffs

Free and Bound variables

Consider a Predicate formula having a part in form of (∃ x) P(x) of (x)P(x), then such part is called x-bound part of the formula. Any occurrence of x in x-bound part is termed as bound occurrence and any occurrence of x which is not x-bound is termed as free occurrence. See the examples below -

- (∃ x) (P(x) ∧ Q(x))
- (∃ x) P(x) ∧ Q(x)

In first example, scope of (∃ x) is (P(x) ∧ Q(x)) and all occurrences of x are bound occurrences. Whereas in second example, scope of (∃ x) is P(x) and last occurrence of x in Q(x) is a free occurrence.

Universe of Discourse

We can limit the class of individuals/objects used in a statment. Here limiting means confining the input variable to a set of particular individuals/objects. Such a restricted class is termed as Universe of Discourse/domain of individual or universe. See the example below:

Some cats are black.

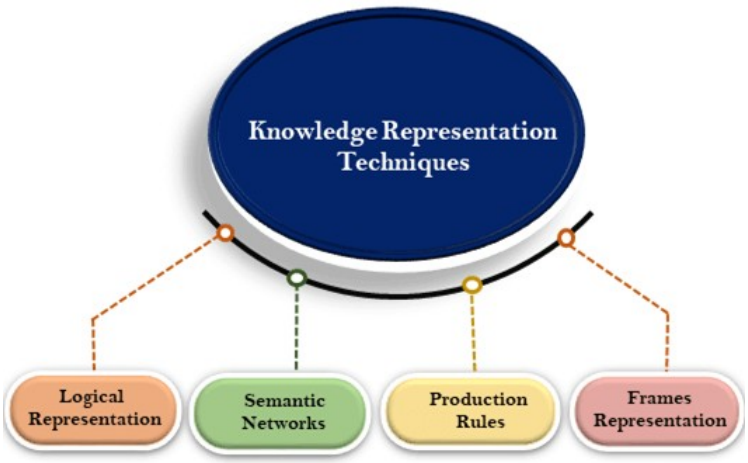
- C(x) : x is a cat.
- B(x) : x is black.
- $(\exists x)(C(x) \wedge B(x))$

If Universe of discourse is E = { Katy, Mille } where katy and Mille are white cats then our third statement is false when we replace x with either Katy or Mille where as if Universe of discourse is E = { Jene, Jackie } where Jene and Jackie black cats then our third statement stands true for Universe of Discourse F.

Techniques of knowledge representation

There are mainly four ways of knowledge representation which are given as follows:

1. Logical Representation
2. Semantic Network Representation
3. Frame Representation
4. Production Rules



1. Logical Representation

Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation. Logical representation means drawing a conclusion based on various conditions. This representation lays down some important communication rules. It consists of precisely defined syntax and semantics which supports the sound inference. Each sentence can be translated into logics using syntax and semantics.

Syntax:

- Syntaxes are the rules which decide how we can construct legal sentences in the logic.
- It determines which symbol we can use in knowledge representation.
- How to write those symbols.

Semantics:

- Semantics are the rules by which we can interpret the sentence in the logic.
- Semantic also involves assigning a meaning to each sentence.

Logical representation can be categorised into mainly two logics:

a. Propositional Logics
b. Predicate logics

Note: We will discuss Prepositional Logics and Predicate logics in later chapters.

Advantages of logical representation:

1. Logical representation enables us to do logical reasoning.
2. Logical representation is the basis for the programming languages.

Disadvantages of logical Representation:

1. Logical representations have some restrictions and are challenging to work with.
2. Logical representation technique may not be very natural, and inference may not be so efficient.

Note: Do not be confused with logical representation and logical reasoning as logical representation is a representation language and reasoning is a process of thinking logically.

2. Semantic Network Representation

Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.

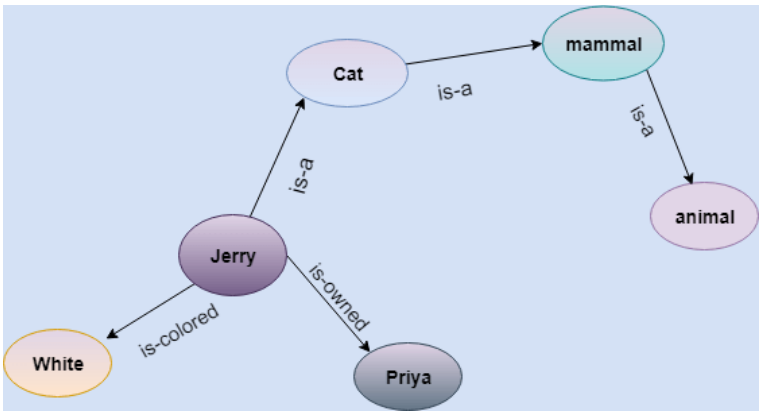
This representation consist of mainly two types of relations:

a. IS-A relation (Inheritance)
b. Kind-of-relation

Example: Following are some statements which we need to represent in the form of nodes and arcs.

Statements:

a. Jerry is a cat.
b. Jerry is a mammal
c. Jerry is owned by Priya.
d. Jerry is brown colored.
e. All Mammals are animal.



In the above diagram, we have represented the different type of knowledge in the form of nodes and arcs. Each object is connected with another object by some relation.

Drawbacks in Semantic representation:

1. Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
2. Semantic networks try to model human-like memory (Which has 1015 neurons and links) to store the information, but in practice, it is not possible to build such a vast semantic network.
3. These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
4. Semantic networks do not have any standard definition for the link names.
5. These networks are not intelligent and depend on the creator of the system.

Advantages of Semantic network:

1. Semantic networks are a natural representation of knowledge.
2. Semantic networks convey meaning in a transparent manner.
3. These networks are simple and easily understandable.

3. Frame Representation

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations. It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called facets.

Facets: The various aspects of a slot is known as **Facets**. Facets are features of frames which enable us to put constraints on the frames. Example: IF-NEEDED facts are called when data of any particular slot is needed. A frame may consist of any number of slots, and a slot may

include any number of facets and facets may have any number of values. A frame is also known as **slot-filter knowledge representation** in artificial intelligence.

Frames are derived from semantic networks and later evolved into our modern-day classes and objects. A single frame is not much useful. Frames system consist of a collection of frames which are connected. In the frame, knowledge about an object or event can be stored together in the knowledge base. The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

Example: 1

Let's take an example of a frame for a book

Slots	Filters
Title	Artificial Intelligence
Genre	Computer Science
Author	Peter Norvig
Edition	Third Edition
Year	1996
Page	1152

Example 2:

Let's suppose we are taking an entity, Peter. Peter is an engineer as a profession, and his age is 25, he lives in city London, and the country is England. So following is the frame representation for this:

Slots	Filter
Name	Peter
Profession	Doctor
Age	25
Marital status	Single
Weight	78

Advantages of frame representation:

1. The frame knowledge representation makes the programming easier by grouping the related data.
2. The frame representation is comparably flexible and used by many applications in AI.
3. It is very easy to add slots for new attribute and relations.
4. It is easy to include default data and to search for missing values.
5. Frame representation is easy to understand and visualize.

Disadvantages of frame representation:

1. In frame system inference mechanism is not be easily processed.
2. Inference mechanism cannot be smoothly proceeded by frame representation.
3. Frame representation has a much generalized approach.

4. Production Rules

Production rules system consist of (**condition, action**) pairs which mean, "If condition then action". It has mainly three parts:

- The set of production rules
- Working Memory
- The recognize-act-cycle

In production rules agent checks for the condition and if the condition exists then production rule fires and corresponding action is carried out. The condition part of the rule determines which rule may be applied to a problem. And the action part carries out the associated problem-solving steps. This complete process is called a recognize-act cycle. The working memory contains the description of the current state of problems-solving and rule can write knowledge to the working memory. This knowledge match and may fire other rules. If there is a new situation (state) generates, then multiple production rules will be fired together, this is called conflict set. In this situation, the agent needs to select a rule from these sets, and it is called a conflict resolution.

Example:

- IF (at bus stop AND bus arrives) THEN action (get into the bus)
- IF (on the bus AND paid AND empty seat) THEN action (sit down).
- IF (on bus AND unpaid) THEN action (pay charges).
- IF (bus arrives at destination) THEN action (get down from the bus).

Advantages of Production rule:

1. The production rules are expressed in natural language.
2. The production rules are highly modular, so we can easily remove, add or modify an individual rule.

Disadvantages of Production rule:

1. Production rule system does not exhibit any learning capabilities, as it does not store the result of the problem for the future uses.
2. During the execution of the program, many rules may be active hence rule-based production systems are inefficient.

CONSTRAINT PROPAGATION

Definition

A constraint satisfaction problem (CSP) is a problem that requires its solution within some limitations/conditions also known as constraints. It consists of the following:

- A finite set of **variables** which stores the solution. (V = {V1, V2, V3,....., Vn})
- A set of **discrete** values known as **domain** from which the solution is picked. (D = {D1, D2, D3,.....,Dn})
- A finite set of **constraints.** (C = {C1, C2, C3,....., Cn})

Please note that the elements in the domain can be both continuous and discrete but in AI, we generally only deal with discrete values. Also, note that all these sets should be finite except for the domain set. Each variable in the variable set can have different domains. For example, consider the Sudoku problem again. Suppose that a row, column and block already

have 3,5 and 7 filled in. Then the domain for all the variables in that row, column and block will be {1,2,4,6,8,9}.

Popular Problems of CSP

The following problems are some of the popular problems that can be solved using CSP:

1. CryptArithmetic (Coding alphabets to numbers.)
2. n-Queen (In an n-queen problem, n queens should be placed in a nXn matrix such that no queen shares the same row, column or diagonal.)
3. Map Coloring (Coloring different regions of map ensuring no adjacent regions have the same color.)
4. Crossword (Everyday puzzles appearing in newspapers.)
5. Sudoku (A number grid.)
6. Latin Square Problem

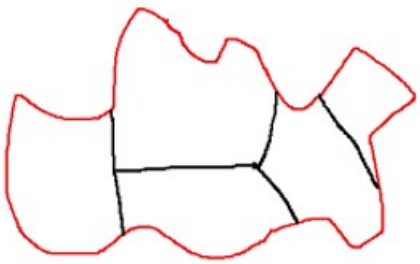
Converting problems to CSPs

A problem to be converted to CSP requires the following steps:

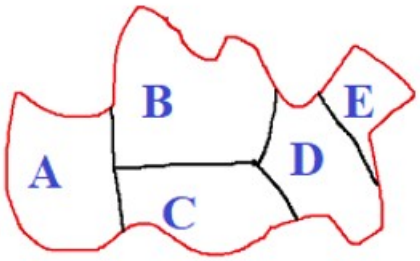
- **Step 1:** Create a variable set.
- **Step 2:** Create a domain set.
- **Step 3:** Create a constraint set with variables and domains (if possible) after considering the constraints.
- **Step 4:** Find an optimal solution.

Example Problem (Map Coloring)

Consider a map coloring problem below. A map is given to you and you have to fill it with only three colors : Blue, Green and Red. But no two adjacent locations should have the same color.



Declare Variable Set, Domain Set and Constraint set



Representing Knowledge using rules in AI

The classic methods of representing knowledge use either rules or logic. Table displays the knowledge for the zoo animals problem in two formats–using rules on the left as implemented within the Knowledge Representation NetLogo model, and using first order logic on the right. Rules are often used in rule-based expert systems, and are either specified explicitly by a knowledge engineer (usually through a process called 'knowledge acquisition' from a human expert), or they are derived from data using a machine learning or data mining algorithm. Rules use a logic-based form for reasoning. Logic is the use of symbolic and mathematical techniques for deductive reasoning, and dates back as a discipline to Aristotle.

Knowledge Representation using rules in Artificial Intelligence

The rule-based method of knowledge representation uses IF-THEN rules (sometimes called conditionaction rules) to specify the knowledge. All the rules for a particular problem form the rules-base, and the knowledge-base comprises three components: the list of rules in the rules-base; the list of known facts in the facts-base; and an inferencing system, which processes the rules to derive new facts via some form of reasoning. A rule consists of an IF part which is a set of conditions (called the *antecedents*) that must be met before the rule is said to 'fire' so that the set of actions in the THEN part (called the *consequents*) are executed. For example, for Rule R1 in Table, if the condition 'animal has hair' is met–that is, there is a known fact in the knowledge base that the animal being classified has hair, then the rule is fired, and the action is to add a further fact 'species is mammal' to the knowledge-base. There may be multiple conditions in the IF part of the rule. For example, Rule R4 has three conditions that must be met before it can be fired. b These conditions are

separated by the AND keyword, and therefore these conditions are called ‘conjunctions’. If they were separated by the OR keyword, they would be called ‘disjunctions’. For the Knowledge Representation NetLogo model, only rules with conjunctions have been implemented. The set of rules and how they are defined for the zoo animals and New Zealand birds problem is shown in NetLogo Code. The third set of rules for the Sailing boats problem can be found by loading the model in NetLogo using the URL link below.

NetLogo Code How the rules are defined for the zoo animals and the New Zealand birds problem in the Knowledge Representation model.

```
if (select-problem = "Zoo animals")
[
setup-rule ;; rule: IF animal has hair
;; THEN species is mammal
[["has hair" "Yes"]]
[["species" "mammal"]]
setup-rule ;; rule: IF animal gives milk
;; THEN species is mammal
[["gives milk" "Yes"]]
[["species" "mammal"]]
setup-rule ;; rule: IF animal eats meat
;; THEN species type is carnivore
[["eats meat" "Yes"]]
[["species type" "carnivore"]]
setup-rule ;; rule: IF animal has pointed teeth AND animal has claws
;; AND animal has forward eyes
;; THEN species type is carnivore
[["has pointed teeth" "Yes"] ["has claws" "Yes"]
["has forward eyes" "Yes"]]
[["species type" "carnivore"]]
setup-rule ;; rule: IF animal is mammal AND animal has hooves
```

```
;; THEN mammal group is ungulate
[["species" "mammal"] ["has hooves" "Yes"]]
[["species type" "ungulate"]]
setup-rule ;; rule: IF species is mammal AND animal chews cud
;; THEN mammal group is ungulate
[["species" "mammal"] ["chews cud" "Yes"]]
[["species type" "ungulate"]]
setup-rule ;; rule: IF species is mammal
;; AND species type is carnivore
;; AND animal has tawny colour
;; AND animal has dark spots
;; THEN animal is cheetah
[["species" "mammal"] ["species type" "carnivore"]
["has tawny colour" "Yes"] ["has dark spots" "Yes"]]
[["animal" "cheetah"]]
setup-rule ;; rule: IF species is mammal
;; AND species type is carnivore
;; AND animal has tawny colour
;; AND animal has black stripes
;; THEN animal is tiger
[["species" "mammal"] ["species type" "carnivore"]
["has tawny colour" "Yes"] ["has black stripes" "Yes"]]
[["animal" "tiger"]]
setup-rule ;; rule: IF species is ungulate AND animal has dark spots
;; AND animal has long neck
;; THEN animal is giraffe
[["species type" "ungulate"] ["has dark spots" "Yes"]
["has long neck" "Yes"]]
[["animal" "giraffe"]]
setup-rule ;; rule: IF species is ungulate
```

```
;; AND animal has black stripes
;; THEN animal is zebra
[["species type" "ungulate"] ["has black stripes" "Yes"]]
[["animal" "zebra"]]
]
if (select-problem = "New Zealand birds")
[
setup-rule ;; rule: IF bird can fly AND bird is a parrot
;; AND bird is alpine
;; THEN bird is a kea
[["can fly" "Yes"] ["parrot" "Yes"] ["alpine" "Yes"]]
[["bird" "Kea" ]]
setup-rule ;; rule: IF bird can fly AND bird is a parrot
;; AND bird is not alpine
;; THEN bird is a kaka
[["can fly" "Yes"] ["parrot" "Yes"] ["alpine" "No"]]
[["bird" "Kaka"]]
setup-rule ;; rule: IF bird can fly AND bird is not a parrot
;; AND bird has white throat
;; THEN bird is a tui
[["can fly" "Yes"] ["parrot" "No"] ["has white throat" "Yes"]]
[["bird" "Tui"]]
setup-rule ;; rule: IF bird can fly AND bird is not a parrot
;; AND bird does not have a white throat
;; THEN bird is a pukeko
[["can fly" "Yes"] ["parrot" "No"] ["has white throat" "No"]]
[["bird" "Pukeko"]]
setup-rule ;; rule: IF bird cannot fly AND bird is an extinct bird
;; AND bird is large
;; THEN bird is a moa
```

[["can fly" "No"] ["extinct" "Yes"] ["large" "Yes"]]

[["bird" "Moa"]]

setup-rule ;; rule: IF bird cannot fly AND bird is extinct

;; AND bird is not large

;; THEN bird is a huia

[["can fly" "No"] ["extinct" "Yes"] ["large" "No"]]

[["bird" "Huia"]]

setup-rule ;; rule: IF bird cannot fly AND bird is not extinct

;; AND bird has long beak

;; THEN bird is a kiwi

[["can fly" "No"] ["extinct" "No"] ["has long beak" "Yes"]]

[["bird" "Kiwi"]]

setup-rule ;; rule: IF bird cannot fly AND bird is not extinct

;; AND bird does not have a long beak

;; THEN bird is a weta

[["can fly" "No"] ["extinct" "No"] ["has long beak" "No"]]

[["bird" "Weta"]]

]

UNIT – 3 (QUESTION BANK)

25	Explain Dempster Shafer theory with the help of suitable example.
26	Explain Symbolic reasoning under uncertainity.
27	Explain Statistical reasoning, and Fuzzy reasoning.
28	Explain non-monotonic reasoning.
29	Explain Baye's rule and its uses.
30	Explain Action, Situation, and Events in Logical engineering.
31	Explain Probabilistic Interference
32	Explain Heuristic method in detail.

UNIT – 3 (NOTES)

Probabilistic reasoning in artificial intelligence

Uncertainty:

The knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty. So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

Causes of uncertainty:

Following are some leading causes of uncertainty to occur in the real world.

1. Information occurred from unreliable sources.
2. Experimental Errors
3. Equipment fault
4. Temperature variation
5. Climate change.

Probabilistic reasoning:

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty. We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance. In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.

- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- **Bayes' rule**
- **Bayesian Statistics**

As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms:

Probability: Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

1. $0 \le P(A) \le 1$, where P(A) is the probability of an event A.
2. $P(A) = 0$, indicates total uncertainty in an event A.
3. $P(A) = 1$, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

$$\text{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- $P(\neg A)$ = probability of a not happening event.
- $P(\neg A) + P(A) = 1$.

Event: Each possible outcome of a variable is called an event.

Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects in the real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional probability:

Conditional probability is a probability of occurring an event when another event has already happened. Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

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

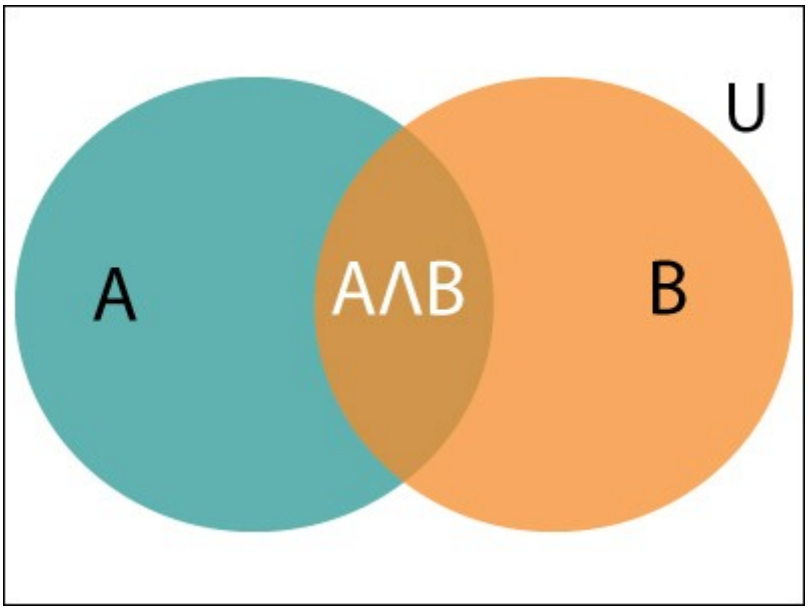
Where P(*A*⋀*B*)= Joint probability of a and B

P(B)= Marginal probability of B.

If the probability of A is given and we need to find the probability of B, then it will be given as:

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

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of **P(A⋀*B*) by P(B)**.



Example:

In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?

Solution:

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

Hence, 57% are the students who like English also like Mathematics.

Bayes' theorem in Artificial intelligence

Bayes' theorem:

Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge. In probability theory, it relates the conditional probability and marginal probabilities of two random events. Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics. It is a way to calculate the value of P(B|A) with the knowledge of P(A|B). Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

Example: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age. Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:

As from product rule we can write:

1. $P(A \wedge B) = P(A|B)\, P(B)$ or

   Similarly, the probability of event B with known event A:

2. $P(A \wedge B) = P(B|A)\, P(A)$

   Equating right hand side of both the equations, we will get:

$$P(A|B) = \frac{P(B|A)\, P(A)}{P(B)} \quad \text{....(a)}$$

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**. It shows the simple relationship between joint and conditional probabilities. Here, P(A|B) is known as **posterior**, which we

need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence. P(A) is called the **prior probability**, probability of hypothesis before considering the evidence. P(B) is called **marginal probability**, pure probability of an evidence.

In the equation (a), in general, we can write P (B) = P(A)*P(B|Ai), hence the Bayes' rule can be written as:

$$P(A_i|B) = \frac{P(A_i)*P(B|A_i)}{\sum_{i=1}^{k} P(A_i)*P(B|A_i)}$$

Where A_1, A_2, A_3,....., A_n is a set of mutually exclusive and exhaustive events.

Applying Bayes' rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A). This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one. Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$\mathbf{P(cause|effect)} = \frac{P(effect|cause)\,P(cause)}{P(effect)}$$

Example-1:

Question: what is the probability that a patient has diseases meningitis with a stiff neck?

Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

- The Known probability that a patient has meningitis disease is 1/30,000.
- The Known probability that a patient has a stiff neck is 2%.

Let a be the proposition that patient has stiff neck and b be the proposition that patient has meningitis. , so we can calculate the following as:

P(a|b) = 0.8

P(b) = 1/30000

P(a)= .02

$$\mathbf{P(b|a)} = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8*(\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

Example-2:

Question: From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

Solution:

$$\mathbf{P(king|face)} = \frac{P(Face|king)*P(King)}{P(Face)} \quad .....(i)$$

P(king): probability that the card is King= 4/52= 1/13

P(face): probability that a card is a face card= 3/13

P(Face|King): probability of face card when we assume it is a king = 1

Putting all values in equation (i) we will get:

$$P(king|face) = \frac{1*(\frac{1}{13})}{(\frac{3}{13})} = \mathbf{1/3,}\ \textbf{it is a probability that a face card is a king card.}$$

Application of Bayes' theorem in Artificial intelligence:

Following are some applications of Bayes' theorem:

- It is used to calculate the next step of the robot when the already executed step is given.
- Bayes' theorem is helpful in weather forecasting.
- It can solve the Monty Hall problem.

Dampster Shafer Theory

Dempster Shafer Theory is given by Arthure P.Dempster in 1967 and his student Glenn Shafer in 1976.

This theory is being released because of following reason:-

- Bayesian theory is only concerned about single evidences.
- Bayesian probability cannot describe ignorance.

DST is an evidence theory, it combines all possible outcomes of the problem. Hence it is used to solve problems where there may be a chance that a different evidence will lead to some different result.

The uncertainity in this model is given by:

1. Consider all possible outcomes.
2. Belief will lead to believe in some possiblity by bringing out some evidence.
3. Plausibility will make evidence compatiblity with possible outcomes.

For e.g:

let us consider a room where four person are presented A, B, C, D(lets say) And suddenly lights out and when the lights come back B has been died due to stabbing in his back with the help of a knife. No one came into the room and no one has leaved the room and B has not committed suicide. Then we have to find out who is the murdrer?

To solve these there are the following possibilities:

- Either {A} or{C} or {D} has killed him.
- Either {A, C} or {C, D} or {A, C} have killed him.
- Or the three of them kill him i.e; {A, C, D}
- None of the kill him {o}(let us say).

These will be the possible evidences by which we can find the murderer by measure of plausibility.

Using the above example we can say :

Set of possible conclusion (P): {p1, p2....pn}

where P is set of possible conclusion and cannot be exhaustive means at least one (p)i must be true.

(p)i must be mutually exclusive.

Power Set will contain 2^n elements where n is number of elements in the possible set.

For eg:-

If P = { a, b, c}, then Power set is given as

{o, {a}, {b}, {c}, {a, b}, {b, c}, {a, c}, {a, b, c}}= 2^3 elements.

Mass function m(K): It is an interpretation of m({K or B}) i.e; it means there is evidence for {K or B} which cannot be divided among more specific beliefs for K and B.

Belief in K: The belief in element K of Power Set is the sum of masses of element which are subsets of K. This can be explained through an example

Lets say K = {a, b, c}

Bel(K) = m(a) + m(b) + m(c) + m(a, b) + m(a, c) + m(b, c) + m(a, b, c)

Plaausiblity in K: It is the sum of masses of set that intersects with K.

i.e; Pl(K) = m(a) + m(b) + m(c) + m(a, b) + m(b, c) + m(a, c) + m(a, b, c)

Characteristics of Dempster Shafer Theory:

- It will ignorance part such that probability of all events aggregate to 1.
- Ignorance is reduced in this theory by adding more and more evidences.
- Combination rule is used to combine various types of possiblities.

Advantages:

- As we add more information, uncertainty interval reduces.
- DST has much lower level of ignorance.
- Diagnose Hierarchies can be represented using this.
- Person dealing with such problems is free to think about evidences.

Disadvantages:

- In this computation effort is high, as we have to deal with 2^n of sets.

Symbolic Reasoning

The basis for intelligent mathematical software is the integration of the "power of symbolic mathematical tools" with the suitable "proof technology".

Mathematical reasoning enjoys a property called monotonic.

"If a conclusion follows from given premises A, B, C, ...

then it also follows from any larger set of premises, as long as the original premises are included."
Human reasoning is not monotonic.

People arrive to conclusions only tentatively, based on partial or incomplete information, reserve the right to retract those conclusions while they learn new facts. Such reasoning is **non-monotonic**, precisely because the set of accepted conclusions have become smaller when the set of premises is expanded.

1. Non-Monotonic Reasoning

Non-Monotonic reasoning is a generic name to a class or a specific theory of reasoning. Non-monotonic reasoning attempts to formalize reasoning with incomplete information by classical logic systems.

The Non-Monotonic reasoning are of the type

- Default reasoning
- Ci umscription
- Truth Maintenance Systems

Default Reasoning

This is a very common form of non-monotonic reasoning. The conclusions are drawn based on what is most likely to be true. There are two approaches, both are logic type, to Default reasoning:

One is Non-monotonic logic and the other is Default logic.

Non-monotonic logic

It has already been defined. It says, "the truth of a proposition may change when new information (axioms) are added and a logic may be build to allows the statement to be retracted."

Non-monotonic logic is predicate logic with one extension called modal operator **M** which means “consistent with everything we know”. The purpose of **M** is to allow consistency.

A way to define consistency with PROLOG notation is : To show that fact **P** is true, we attempt to prove **¬P**.
If we fail we may say that **P** is consistent since **¬P** is false.

Example:

$\forall$ x : plays_instrument(x) $\wedge$ M manage(x) $\rightarrow$ jazz_musician(x)

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

■ **Default Logic**

Default logic initiates a new inference rule: $\frac{A : B}{C}$ where

- **A** is known as the prerequisite,
- **B** as the justification, and
- **C** as the consequent.

Read the above inference rule as:
" if A, and if it is consistent with the rest of what is known to assume that B, then conclude that C ".
The rule says that given the prerequisite, the consequent can be inferred, provided it is consistent with the rest of the data.

‡ Example : Rule that "birds typically fly" would be represented as

$$\frac{bird(x) : flies(x)}{flies(x)}$$ which says

" If **x** is a bird andthe claim that **x** flies is consistent with what we know, then infer that **x** flies".

‡ Note : Since, all we know about Tweety is that :

Tweety is a bird, we therefore inferred that Tweety flies.

The idea behind non-monotonic reasoning is to reason with first order logic, and if an inference can not be obtained then use the set of default rules available within the first order formulation.

‡ Applying Default Rules:

While applying default rules, it is necessary to check their justifications for consistency, not only with initial data, but also with the consequents of any other default rules that may be applied. The application of one rule may thus block the application of another. To solve this problem, the concept of default theory was extended.

‡ Default Theory

It consists of a set of premises **W** and a set of default rules **D**.
An extension for a default theory is a set of sentences **E** which can be derived from **W** by applying as many rules of **D** as possible (together with the rules of deductive inference) without generating inconsistency.
Note : **D** the set of default rules has a unique syntax of the form

$$\frac{\alpha(\vec{x}) : E\,\beta(\vec{x})}{\gamma(\vec{x})}$$ where

$\alpha(\vec{x})$ is the prerequisite of the default rule

$E\,\beta(\vec{x})$ is the consistency test of the default rule

$\gamma(\vec{x})$ is the consequent of the default rule

The rule can be read as

For all individual $x_1 \ldots x_m$

If $\alpha(\vec{x})$ is believed and

If each of $\beta(\vec{x})$ is consistent with our beliefs,

Then $\gamma(\vec{x})$ may be believed.

Example:

A Default Rule says " *Typically an American adult owns a car* ".

$$\frac{\text{American}(x) \wedge \text{Adult}(x) : M((\exists y) . \text{car}(y) \wedge \text{owns}(x,y))}{((\exists y) . \text{car}(y) \wedge \text{owns}(x,y))}$$

The rule is explained below :

The rule is only accessed if we wish to know whether or not John owns a car then an answer cannot be deduced from our current beliefs. This default rule is applicable if we can prove from our beliefs that John is an American and an adult, and believing that there is some car that is owned by John does not lead to an inconsistency. If these two sets of premises are satisfied, then the rule states that we can conclude that John owns a car.

Ci umscription

Ci umscription is a non-monotonic logic to formalize the common sense assumption. Ci umscription is a formalized rule of conjecture (guess) that can be used along with the rules of inference of first order logic.

Ci umscription involves formulating rules of thumb with "abnormality" predicates and then restricting the extension of these predicates,

ci umscribing them, so that they apply to only those things to which they are currently known.

Example: Take the case of Bird Tweety

The rule of thumb is that "birds typically fly" is conditional. The predicate "Abnormal" signifies abnormality with respect to flying ability.

Observe that the rule $\forall$ **x(Bird(x) & ¬ Abnormal(x)** $\rightarrow$ **Flies))** does not allow us to infer that *"Tweety flies"*, since we do not know that he is abnormal with respect to flying ability.

But if we add axioms which ci umscribe the *abnormality predicate* to which they are currently known say *"Bird Tweety*" then the inference can be drawn. This inference is non-monotonic.

Truth Maintenance Systems

Reasoning Maintenance System (RMS) is a critical part of a reasoning system. Its purpose is to assure that inferences made by the reasoning system (RS) are valid.

The RS provides the RMS with information about each inference it performs, and in return the RMS provides the RS with information about the whole set of inferences. Several implementations of RMS have been proposed for non-monotonic reasoning. The important ones are the:

Truth Maintenance Systems (TMS) and Assumption-based Truth Maintenance Systems (ATMS). The TMS maintains the consistency of a knowledge base as soon as new knowledge is added. It considers only one state at a time so it is not possible to manipulate environment. The ATMS is intended to maintain multiple environments. The typical functions of TMS are presented in the next slide.

Truth Maintenance Systems (TMS)

A truth maintenance system maintains consistency in knowledge representation of a knowledge base. The functions of TMS are to:

- **Provide justifications for conclusions**

Statistical Reasoning:

In the logic based approaches described, we have assumed that everything is either believed false or believed true. However, it is often useful to represent the fact that we believe such that something is probably true, or true with probability (say) 0.65. This is useful for dealing with problems where there is randomness and unpredictability (such as in games of chance) and also for dealing with problems where we could, if we had sufficient information, work out exactly what is true. To do all this in a principled way requires techniques for probabilistic reasoning. In this section, the Bayesian Probability Theory is first described and then discussed how uncertainties are treated.

Recall glossary of terms

■ **Probabilities:**

Usually, are descriptions of the likelihood of some event occurring (ranging from **0** to **1**).

■ **Event:**

One or more outcomes of a probability experiment.

■ **Probability Experiment:**

Process which leads to well-defined results call outcomes.

■ **Sample Space:**

Set of all possible outcomes of a probability experiment.

■ **Independent Events:**

Two events, E1 and E2, are independent if the fact that E1 occurs does not affect the probability of E2 occurring.

■ **Mutually Exclusive Events:**

Events E1, E2, ..., En are said to be mutually exclusive if the occurrence of any one of them automatically implies the non-occurrence of the remaining n − 1 events.

■ **Disjoint Events:**

Another name for mutually exclusive events.

■ **Classical Probability:**

Also called a priori theory of probability. The probability of event **A** = no of possible outcomes **f** divided by the total no of possible outcomes **n** ; ie., **P(A) = f / n.**

Assumption: All possible outcomes are equal likely.

■ **Empirical Probability:**

Determined analytically, using knowledge about the nature of the experiment rather than through actual experimentation.

■ **Conditional Probability:**

The probability of some event **A**, given the occurrence of some other event **B**. Conditional probability is written **P(A|B)**, and read as "the probability of **A**, given **B** ".

■ **Joint probability:**

The probability of two events in conjunction. It is the probability of both events together. The joint probability of **A** and **B** is written **P(A** ∩; also written as **P(A, B)**.

Marginal Probability:

The probability of one event, regardless of the other event. The marginal probability of **A** is written **P(A)**, and the marginal probability of **B** is written **P(B)**.

Examples

Sample Space - Rolling two dice

The sums can be **{ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 }.**

Note that each of these are not equally likely. The only way to get a sum **2** is to roll a **1** on both dice, but can get a sum **4** by rolling out comes as **(1,3), (2,2), or (3,1)**.

Table below illustrates a sample space for the sum obtain.

			Second Dice			
First dice	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Classical Probability

Table below illustrates frequency and distribution for the above sums.

Sum	2	3	4	5	6	7	8	9	10	11	12
Frequency	1	2	3	4	5	6	5	4	3	2	1
Relative frequency	1/36	2/36	3/36	4/36	5/36	6/36	5/36	4/36	3/36	1/36	1/36

The classical probability is the relative frequency of each event. Classical probability **P(E) = n(E) / n(S)**; **P(6) = 5 / 36, P(8) = 5 / 36**

Empirical Probability

The empirical probability of an event is the relative frequency of a frequency distribution based upon observation **P(E) = f / n**

What is Fuzzy Logic?

Fuzzy Logic (FL) is a method of reasoning that resembles human reasoning. The approach of FL imitates the way of decision making in humans that involves all intermediate possibilities between digital values YES and NO. The conventional logic block that a computer can understand takes precise input and produces a definite output as TRUE or FALSE, which is equivalent to human's YES or NO. The inventor of fuzzy logic, Lotfi Zadeh, observed that unlike computers, the human decision making includes a range of possibilities between YES and NO, such as –

CERTAINLY YES
POSSIBLY YES
CANNOT SAY
POSSIBLY NO
CERTAINLY NO

The fuzzy logic works on the levels of possibilities of input to achieve the definite output.

Implementation

- It can be implemented in systems with various sizes and capabilities ranging from small micro-controllers to large, networked, workstation-based control systems.
- It can be implemented in hardware, software, or a combination of both.

Why Fuzzy Logic?

Fuzzy logic is useful for commercial and practical purposes.

- It can control machines and consumer products.
- It may not give accurate reasoning, but acceptable reasoning.
- Fuzzy logic helps to deal with the uncertainty in engineering.

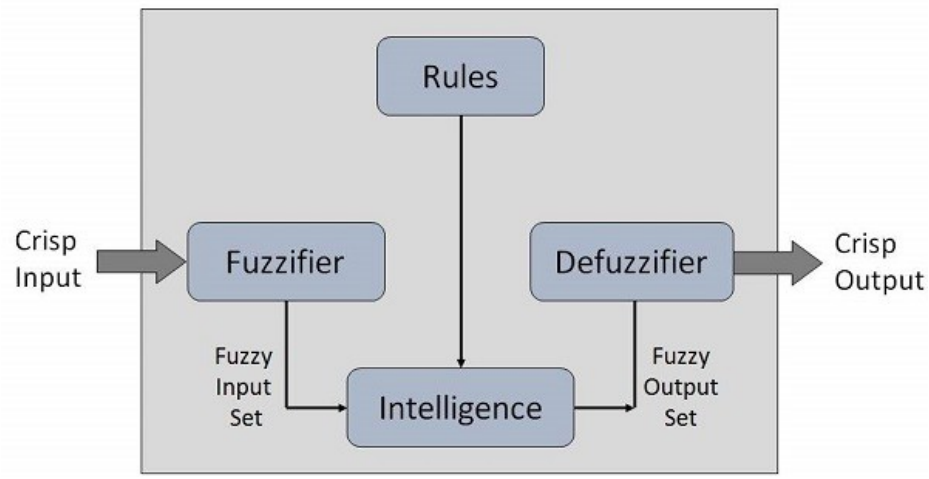
Fuzzy Logic Systems Architecture

It has four main parts as shown –

- **Fuzzification Module** – It transforms the system inputs, which are crisp numbers, into fuzzy sets. It splits the input signal into five steps such as –

LP	x is Large Positive
MP	x is Medium Positive
S	x is Small
MN	x is Medium Negative
LN	x is Large Negative

- **Knowledge Base** – It stores IF-THEN rules provided by experts.
- **Inference Engine** – It simulates the human reasoning process by making fuzzy inference on the inputs and IF-THEN rules.
- **Defuzzification Module** – It transforms the fuzzy set obtained by the inference engine into a crisp value.



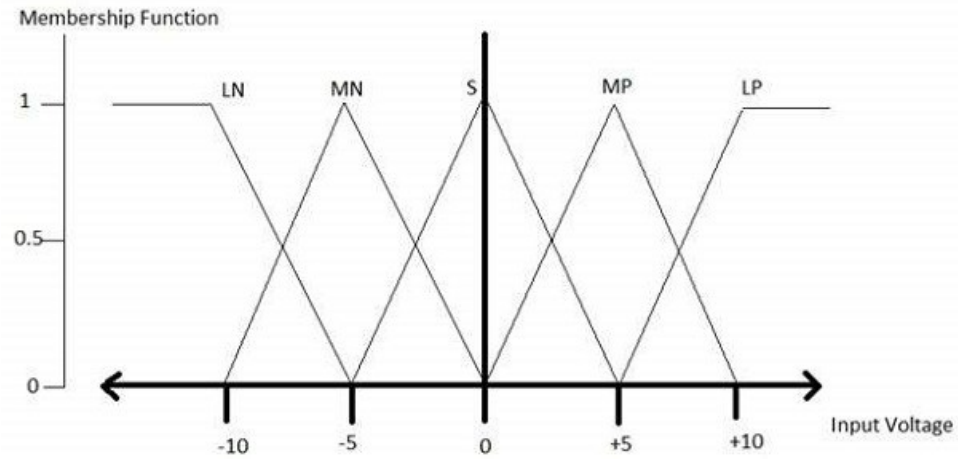
The membership functions work on fuzzy sets of variables.

Membership Function

Membership functions allow you to quantify linguistic term and represent a fuzzy set graphically. A membership function for a fuzzy *set A* on the universe of discourse X is defined as μ_A:X → [0,1]. Here, each element of *X* is mapped to a value between 0 and 1. It is called membership value or degree of membership. It quantifies the degree of membership of the element in *X* to the fuzzy set *A*.

- x axis represents the universe of discourse.
- y axis represents the degrees of membership in the [0, 1] interval.

There can be multiple membership functions applicable to fuzzify a numerical value. Simple membership functions are used as use of complex functions does not add more precision in the output. All membership functions for LP, MP, S, MN, and LN are shown as below –



The triangular membership function shapes are most common among various other membership function shapes such as trapezoidal, singleton, and Gaussian. Here, the input to 5-level fuzzifier varies from -10 volts to +10 volts. Hence the corresponding output also changes.

Non-monotonic Reasoning

In Non-monotonic reasoning, some conclusions may be invalidated if we add some more information to our knowledge base. Logic will be said as non-monotonic if some conclusions

can be invalidated by adding more knowledge into our knowledge base. Non-monotonic reasoning deals with incomplete and uncertain models. "Human perceptions for various things in daily life, "is a general example of non-monotonic reasoning. **Example:** Let suppose the knowledge base contains the following knowledge:

- Birds can fly
- Penguins cannot fly
- Pitty is a bird

So from the above sentences, we can conclude that Pitty can fly**.** However, if we add one another sentence into knowledge base "Pitty is a penguin", which concludes "Pitty cannot fly", so it invalidates the above conclusion.

Advantages of Non-monotonic reasoning:

- For real-world systems such as Robot navigation, we can use non-monotonic reasoning.
- In Non-monotonic reasoning, we can choose probabilistic facts or can make assumptions.

Disadvantages of Non-monotonic Reasoning:

- In non-monotonic reasoning, the old facts may be invalidated by adding new sentences.
- It cannot be used for theorem proving**.**

UNIT – 4 (QUESTION BANK)

33	What is an Expert system? What are its characteristics? Explain.
34	Write short notes: (a) NLP (b) AI Applications on Robotics.
35	Explain Rule based architecture.
36	Define the concept of knowledge acquisition in detail.

37	What are the various current trends in the Intelligent systems?
38	What are the difference between forward chaining and backward chaining?
39	What are the various components of Robot?
40	Explain the applications of Robotics?

UNIT – 4 (NOTES)

NATURAL LANGUAGE PROCESSING

Natural Language Processing (NLP) refers to AI method of communicating with an intelligent systems using a natural language such as English. Processing of Natural Language is required when you want an intelligent system like robot to perform as per your instructions, when you want to hear decision from a dialogue based clinical expert system, etc. The field of NLP involves making computers to perform useful tasks with the natural languages humans use. The input and output of an NLP system can be;

- Speech
- Written Text

Components of NLP

There are two components of NLP as given in Natural Language Understanding (NLU) involves the following tasks are;

- Mapping the given input in natural language into useful representations.
- Analyzing different aspects of the language.

Natural Language Generation (NLG)

It is the process of producing meaningful phrases and sentences in the form of natural language from some internal representation. It involves;

- **Text planning** – It includes retrieving the relevant content from knowledge base.
- **Sentence planning** – It includes choosing required words, forming meaningful phrases, setting tone of the sentence.

- **Text Realization** – It is mapping sentence plan into sentence structure.

The NLU is harder than NLG. Difficulties in NLU is that NL has an extremely rich form and structure. It is very ambiguous. There can be different levels of ambiguity –

- **Lexical ambiguity** – It is at very primitive level such as word-level.
- For example, treating the word "board" as noun or verb?
- **Syntax Level ambiguity** – A sentence can be parsed in different ways.
- For example, "He lifted the beetle with red cap." – Did he use cap to lift the beetle or he lifted a beetle that had red cap?
- **Referential ambiguity** – Referring to something using pronouns. For example, Rima went to Gauri. She said, "I am tired." – Exactly who is tired?
- One input can mean different meanings.
- Many inputs can mean the same thing.

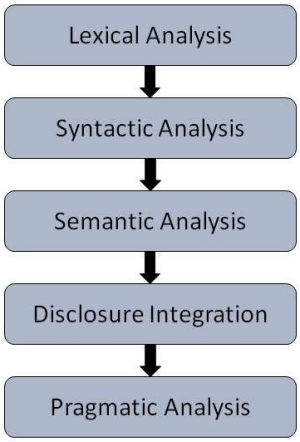
NLP Terminology

- **Phonology** – It is study of organizing sound systematically.
- **Morphology** – It is a study of construction of words from primitive meaningful units.
- **Morpheme** – It is primitive unit of meaning in a language.
- **Syntax** – It refers to arranging words to make a sentence. It also involves determining the structural role of words in the sentence and in phrases.
- **Semantics** – It is concerned with the meaning of words and how to combine words into meaningful phrases and sentences.
- **Pragmatics** – It deals with using and understanding sentences in different situations and how the interpretation of the sentence is affected.
- **Discourse** – It deals with how the immediately preceding sentence can affect the interpretation of the next sentence.
- **World Knowledge** – It includes the general knowledge about the world.

Steps in NLP

There are general five steps –

- **Lexical Analysis** – It involves identifying and analyzing the structure of words. Lexicon of a language means the collection of words and phrases in a language. Lexical analysis is dividing the whole chunk of txt into paragraphs, sentences, and words.
- **Syntactic Analysis (Parsing)** – It involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words. The sentence such as "The school goes to boy" is rejected by English syntactic analyzer.



- **Semantic Analysis** – It draws the exact meaning or the dictionary meaning from the text. The text is checked for meaningfulness. It is done by mapping syntactic structures and objects in the task domain. The semantic analyzer disregards sentence such as "hot ice-cream".
- **Discourse Integration** – The meaning of any sentence depends upon the meaning of the sentence just before it. In addition, it also brings about the meaning of immediately succeeding sentence.
- **Pragmatic Analysis** – During this, what was said is re-interpreted on what it actually meant. It involves deriving those aspects of language which require real world knowledge.

Implementation Aspects of Syntactic Analysis

There are a number of algorithms researchers have developed for syntactic analysis, but we consider only the following simple methods are;

- Context-Free Grammar
- Top-Down Parser

Context-Free Grammar

It is the grammar that consists rules with a single symbol on the left-hand side of the rewrite rules. Let us create grammar to parse a sentence –

"The bird pecks the grains"

Articles (DET) – a | an | the

Nouns – bird | birds | grain | grains

Noun Phrase (NP) – Article + Noun | Article + Adjective + Noun

= DET N | DET ADJ N

Verbs – pecks | pecking | pecked

Verb Phrase (VP) – NP V | V NP

Adjectives (ADJ) – beautiful | small | chirping

The parse tree breaks down the sentence into structured parts so that the computer can easily understand and process it. In order for the parsing algorithm to construct this parse tree, a set of rewrite rules, which describe what tree structures are legal, need to be constructed.

These rules say that a certain symbol may be expanded in the tree by a sequence of other symbols. According to first order logic rule, if there are two strings Noun Phrase (NP) and Verb Phrase (VP), then the string combined by NP followed by VP is a sentence. The rewrite rules for the sentence are as follows –

S → NP VP

NP → DET N | DET ADJ N

VP → V NP

Lexocon –

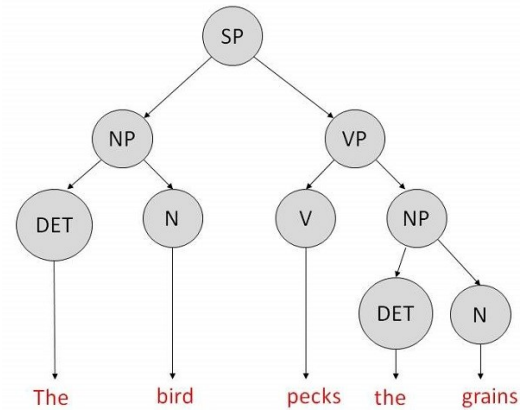
DET → a | the

ADJ → beautiful | perching

N → bird | birds | grain | grains

V → peck | pecks | pecking

The parse tree can be created as shown –



Now, consider the above rewrite rules. Since V can be replaced by both, "peck" or "pecks", sentences such as "The bird peck the grains" can be wrongly permitted. i. e. the subject-verb agreement error is approved as correct.

Merit – The simplest style of grammar, therefore widely used one.

Demerits –

- They are not highly precise. For example, “The grains peck the bird”, is a syntactically correct according to parser, but even if it makes no sense, parser takes it as a correct sentence.
- To bring out high precision, multiple sets of grammar need to be prepared. It may require a completely different sets of rules for parsing singular and plural variations, passive sentences, etc., which can lead to creation of huge set of rules that are unmanageable.

Top-Down Parser

Here, the parser starts with the S symbol and attempts to rewrite it into a sequence of terminal symbols that matches the classes of the words in the input sentence until it consists entirely of terminal symbols. These are then checked with the input sentence to see if it matched. If not, the process is started over again with a different set of rules. This is repeated until a specific rule is found which describes the structure of the sentence.

Merit – It is simple to implement.

Demerits –

- It is inefficient, as the search process has to be repeated if an error occurs.
- Slow speed of working.

What are Expert Systems?

The expert systems are the computer applications developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise. Characteristics of Expert Systems are;

- High performance
- Understandable
- Reliable
- Highly responsive

Capabilities of Expert Systems: The expert systems are capable of –

- Advising
- Instructing and assisting human in decision making
- Demonstrating
- Deriving a solution
- Diagnosing
- Explaining
- Interpreting input
- Predicting results
- Justifying the conclusion
- Suggesting alternative options to a problem

They are incapable of –

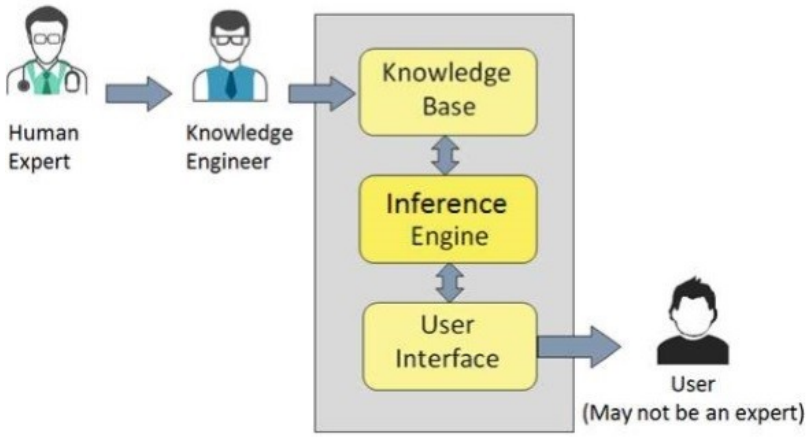
- Substituting human decision makers
- Possessing human capabilities
- Producing accurate output for inadequate knowledge base
- Refining their own knowledge

Components of Expert Systems:

The components of ES include –

- Knowledge Base
- Inference Engine
- User Interface

Let us see them one by one briefly –



Knowledge Base

It contains domain-specific and high-quality knowledge. Knowledge is required to exhibit intelligence. The success of any ES majorly depends upon the collection of highly accurate and precise knowledge.

What is Knowledge?

The data is collection of facts. The information is organized as data and facts about the task domain. Data, information, and past experience combined together are termed as knowledge.

Components of Knowledge Base

The knowledge base of an ES is a store of both, factual and heuristic knowledge.

- **Factual Knowledge** – It is the information widely accepted by the Knowledge Engineers and scholars in the task domain.
- **Heuristic Knowledge** – It is about practice, accurate judgement, one's ability of evaluation, and guessing.

Knowledge representation

It is the method used to organize and formalize the knowledge in the knowledge base. It is in the form of IF-THEN-ELSE rules.

Knowledge Acquisition

The success of any expert system majorly depends on the quality, completeness, and accuracy of the information stored in the knowledge base. The knowledge base is formed by readings from various experts, scholars, and the Knowledge Engineers. The knowledge engineer is a person with the qualities of empathy, quick learning, and case analyzing skills. He acquires information from subject expert by recording, interviewing, and observing him at work, etc. He then categorizes and organizes the information in a meaningful way, in the form of IF-THEN-ELSE rules, to be used by interference machine. The knowledge engineer also monitors the development of the ES.

Inference Engine

Use of efficient procedures and rules by the Inference Engine is essential in deducting a correct, flawless solution. In case of knowledge-based ES, the Inference Engine acquires and manipulates the knowledge from the knowledge base to arrive at a particular solution. In case of rule based ES, it –

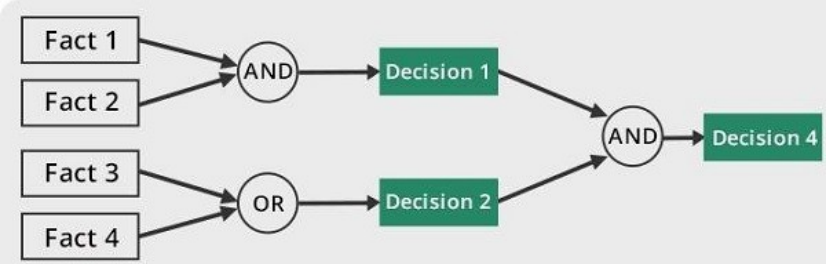
- Applies rules repeatedly to the facts, which are obtained from earlier rule application.
- Adds new knowledge into the knowledge base if required.
- Resolves rules conflict when multiple rules are applicable to a particular case.

To recommend a solution, the Inference Engine uses the following strategies –

- Forward Chaining
- Backward Chaining

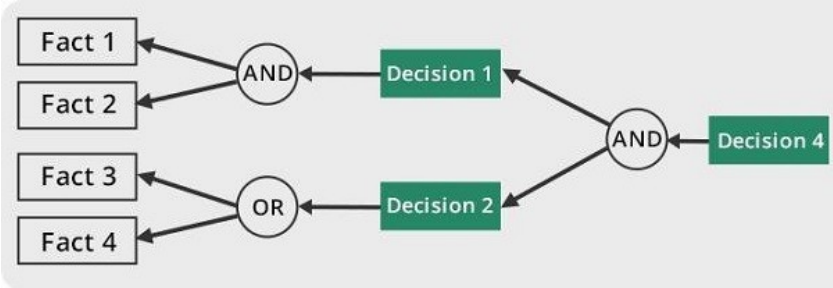
Forward Chaining

The Inference Engine follows the chain of conditions and derivations and finally deduces the outcome. It considers all the facts and rules, and sorts them before concluding to a solution. This strategy is followed for working on conclusion, result, or effect. For example, prediction of share market status as an effect of changes in interest rates.



Backward Chaining

On the basis of what has already happened, the Inference Engine tries to find out which conditions could have happened in the past for this result. This strategy is followed for finding out cause or reason. For example, diagnosis of blood cancer in humans.



User Interface

User interface provides interaction between user of the ES and the ES itself. It is generally Natural Language Processing so as to be used by the user who is well-versed in the task domain. The user of the ES need not be necessarily an expert in Artificial Intelligence.

It explains how the ES has arrived at a particular recommendation. The explanation may appear in the following forms –

- Natural language displayed on screen.
- Verbal narrations in natural language.
- Listing of rule numbers displayed on the screen.

The user interface makes it easy to trace the credibility of the deductions.

Requirements of Efficient ES User Interface

- It should help users to accomplish their goals in shortest possible way.
- It should be designed to work for user's existing or desired work practices.

- Its technology should be adaptable to user's requirements; not the other way round.
- It should make efficient use of user input.

Expert Systems Limitations

No technology can offer easy and complete solution. Large systems are costly, require significant development time, and computer resources. ESs have their limitations which include –

- Limitations of the technology
- Difficult knowledge acquisition
- ES are difficult to maintain
- High development costs

Applications of Expert System

The following table shows where ES can be applied.

Application	Description
Design Domain	Camera lens design, automobile design.
Medical Domain	Diagnosis Systems to deduce cause of disease from observed data, conduction medical operations on humans.
Monitoring Systems	Comparing data continuously with observed system or with prescribed behavior such as leakage monitoring in long petroleum pipeline.
Process Control Systems	Controlling a physical process based on monitoring.
Knowledge Domain	Finding out faults in vehicles, computers.
Finance/Commerce	Detection of possible fraud, suspicious transactions, stock market trading, Airline scheduling, cargo scheduling.

Expert System Technology

There are several levels of ES technologies available. Expert systems technologies include –

- **Expert System Development Environment** – The ES development environment includes hardware and tools. They are;
    - Workstations, minicomputers, mainframes.
    - High level Symbolic Programming Languages such as **LIS**t **P**rogramming (LISP) and **PRO**grammation en **LOG**ique (PROLOG).
    - Large databases.
- **Tools** – They reduce the effort and cost involved in developing an expert system to large extent.
    - Powerful editors and debugging tools with multi-windows.
    - They provide rapid prototyping
    - Have Inbuilt definitions of model, knowledge representation, and inference design.
- **Shells** – A shell is nothing but an expert system without knowledge base. A shell provides the developers with knowledge acquisition, inference engine, user interface, and explanation facility. For example, few shells are given below –
    - Java Expert System Shell (JESS) that provides fully developed Java API for creating an expert system.
    - Vidwan, a shell developed at the National Centre for Software Technology, Mumbai in 1993. It enables knowledge encoding in the form of IF-THEN rules.

Development of Expert Systems: General Steps

The process of ES development is iterative. Steps in developing the ES include –

Identify Problem Domain

- The problem must be suitable for an expert system to solve it.
- Find the experts in task domain for the ES project.
- Establish cost-effectiveness of the system.

Design the System

- Identify the ES Technology
- Know and establish the degree of integration with the other systems and databases.

- Realize how the concepts can represent the domain knowledge best.

Develop the Prototype

From Knowledge Base: The knowledge engineer works to –

- Acquire domain knowledge from the expert.
- Represent it in the form of If-THEN-ELSE rules.

Test and Refine the Prototype

- The knowledge engineer uses sample cases to test the prototype for any deficiencies in performance.
- End users test the prototypes of the ES.

Develop and Complete the ES

- Test and ensure the interaction of the ES with all elements of its environment, including end users, databases, and other information systems.
- Document the ES project well.
- Train the user to use ES.

Maintain the System

- Keep the knowledge base up-to-date by regular review and update.
- Cater for new interfaces with other information systems, as those systems evolve.

Benefits of Expert Systems

- **Availability** – They are easily available due to mass production of software.
- **Less Production Cost** – Production cost is reasonable. This makes them affordable.
- **Speed** – They offer great speed. They reduce the amount of work an individual puts in.
- **Less Error Rate** – Error rate is low as compared to human errors.
- **Reducing Risk** – They can work in the environment dangerous to humans.
- **Steady response** – They work steadily without getting motional, tensed or fatigued.

What are Robots?

Robots are the artificial agents acting in real world environment.

Objective

Robots are aimed at manipulating the objects by perceiving, picking, moving, modifying the physical properties of object, destroying it, or to have an effect thereby freeing manpower from doing repetitive functions without getting bored, distracted, or exhausted.

What is Robotics?

Robotics is a branch of AI, which is composed of Electrical Engineering, Mechanical Engineering, and Computer Science for designing, construction, and application of robots.

Aspects of Robotics

- The robots have mechanical construction, form, or shape designed to accomplish a particular task.
- They have **electrical components** which power and control the machinery.
- They contain some level of **computer program** that determines what, when and how a robot does something.

Difference in Robot System and Other AI Program

Here is the difference between the two –

AI Programs	Robots
They usually operate in computer-stimulated worlds.	They operate in real physical world
The input to an AI program is in symbols and rules.	Inputs to robots is analog signal in the form of speech waveform or images
They need general purpose computers to operate on.	They need special hardware with sensors and effectors.

Robot Locomotion

Locomotion is the mechanism that makes a robot capable of moving in its environment. There are various types of locomotion –

- Legged
- Wheeled
- Combination of Legged and Wheeled Locomotion

- Tracked slip/skid

Legged Locomotion

- This type of locomotion consumes more power while demonstrating walk, jump, trot, hop, climb up or down, etc.
- It requires more number of motors to accomplish a movement. It is suited for rough as well as smooth terrain where irregular or too smooth surface makes it consume more power for a wheeled locomotion. It is little difficult to implement because of stability issues.
- It comes with the variety of one, two, four, and six legs. If a robot has multiple legs then leg coordination is necessary for locomotion.

The total number of possible **gaits** (a periodic sequence of lift and release events for each of the total legs) a robot can travel depends upon the number of its legs.

If a robot has k legs, then the number of possible events N = (2k-1)!.

In case of a two-legged robot (k=2), the number of possible events is N = (2k-1)! = (2*2-1)! = 3! = 6.

Hence there are six possible different events –

- Lifting the Left leg
- Releasing the Left leg
- Lifting the Right leg
- Releasing the Right leg
- Lifting both the legs together
- Releasing both the legs together

In case of k=6 legs, there are 39916800 possible events. Hence the complexity of robots is directly proportional to the number of legs.

Wheeled Locomotion

It requires fewer number of motors to accomplish a movement. It is little easy to implement as there are less stability issues in case of more number of wheels. It is power efficient as compared to legged locomotion.

- **Standard wheel** – Rotates around the wheel axle and around the contact
- **Castor wheel** – Rotates around the wheel axle and the offset steering joint.
- **Swedish 45° and Swedish 90° wheels** – Omni-wheel, rotates around the contact point, around the wheel axle, and around the rollers.
- **Ball or spherical wheel** – Omnidirectional wheel, technically difficult to implement.

Slip/Skid Locomotion

In this type, the vehicles use tracks as in a tank. The robot is steered by moving the tracks with different speeds in the same or opposite direction. It offers stability because of large contact area of track and ground.

Components of a Robot

Robots are constructed with the following are;

- **Power Supply** – The robots are powered by batteries, solar power, hydraulic, or pneumatic power sources.
- **Actuators** – They convert energy into movement.
- **Electric motors (AC/DC)** – They are required for rotational movement.
- **Pneumatic Air Muscles** – They contract almost 40% when air is sucked in them.
- **Muscle Wires** – They contract by 5% when electric current is passed through them.
- **Piezo Motors and Ultrasonic Motors** – Best for industrial robots.
- **Sensors** – They provide knowledge of real time information on the task environment. Robots are equipped with vision sensors to be to compute the depth in the environment. A tactile sensor imitates the mechanical properties of touch receptors of human fingertips.

Computer Vision

This is a technology of AI with which the robots can see. The computer vision plays vital role in the domains of safety, security, health, access, and entertainment. Computer vision automatically extracts, analyzes, and comprehends useful information from a single image or an array of images. This process involves development of algorithms to accomplish automatic visual comprehension.

Hardware of Computer Vision System

This involves –

- Power supply
- Image acquisition device such as camera
- A processor
- A software
- A display device for monitoring the system
- Accessories such as camera stands, cables, and connectors

Tasks of Computer Vision

- **OCR** – In the domain of computers, Optical Character Reader, a software to convert scanned documents into editable text, which accompanies a scanner.

- **Face Detection** – Many state-of-the-art cameras come with this feature, which enables to read the face and take the picture of that perfect expression. It is used to let a user access the software on correct match.
- **Object Recognition** – They are installed in supermarkets, cameras, high-end cars such as BMW, GM, and Volvo.
- **Estimating Position** – It is estimating position of an object with respect to camera as in position of tumor in human's body.

Application Domains of Computer Vision

- Agriculture
- Autonomous vehicles
- Biometrics
- Character recognition
- Forensics, security, and surveillance
- Industrial quality inspection
- Face recognition
- Gesture analysis
- Geoscience
- Medical imagery
- Pollution monitoring
- Process control
- Remote sensing
- Robotics
- Transport

Applications of Robotics

The robotics has been instrumental in the various domains such as –

- **Industries** – Robots are used for handling material, cutting, welding, color coating, drilling, polishing, etc.
- **Military** – Autonomous robots can reach inaccessible and hazardous zones during war. A robot named Daksh, developed by Defense Research and Development Organization (DRDO), is in function to destroy life-threatening objects safely.

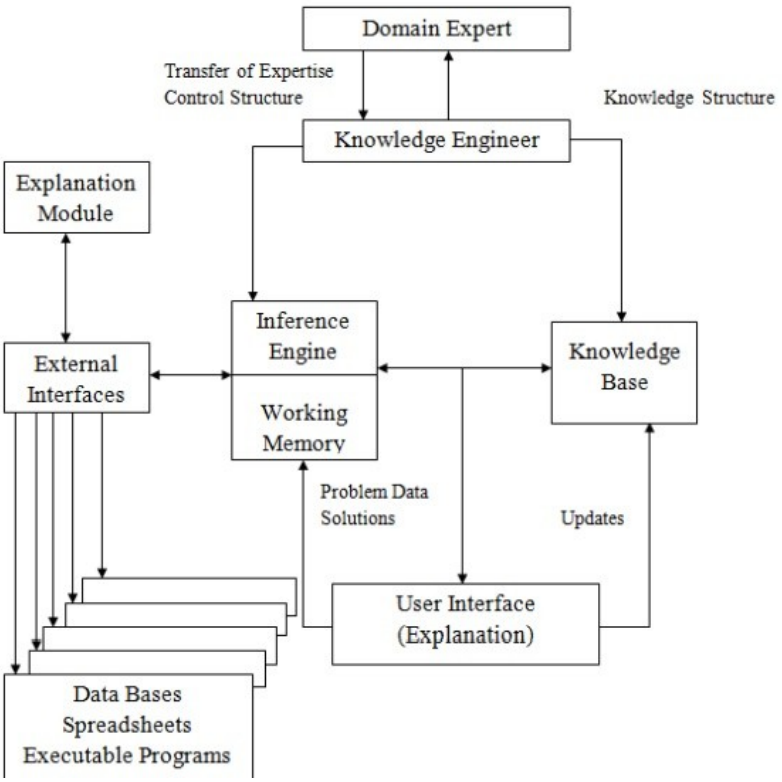
- **Medicine** – The robots are capable of carrying out hundreds of clinical tests simultaneously, rehabilitating permanently disabled people, and performing complex surgeries such as brain tumors.
- **Exploration** – The robot rock climbers used for space exploration, underwater drones used for ocean exploration are to name a few.
- **Entertainment** – Disney's engineers have created hundreds of robots for movie making.

RULE BASED ARCHITECTURE OF AN EXPERT SYSTEM

The most common form of architecture used in expert and other types of knowledge based systems is the production system or it is called rule based systems. This type of system uses knowledge encoded in the form of production rules i.e. if-then rules. The rule has a conditional part on the left hand side and a conclusion or action part on the right hand side. For example if: condition1 and condition2 and condition3

Then: Take action4

Each rule represents a small chunk of knowledge to the given domain of expertise. When the known facts support the conditions in the rule's left side, the conclusion or action part of the rule is then accepted as known. The rule based architecture of an expert system consists of the domain expert, knowledge engineer, inference engine, working memory, knowledge base, external interfaces, user interface, explanation module, database spreadsheets executable programs s mentioned in figure.



Integration of Expert systems Components

The components of the rule based architecture are as follows.

1. **User Interface:** It is the mechanism by which the user and the expert system communicate with each other i.e. the use interacts with the system through a user interface. It acts as a bridge between user and expert system. This module accepts the user queries and submits those to the expert system. The user normally consults the expert system for following reasons.

   a) To get answer of his/her queries.

   b) To get explanation about the solution for psychological satisfaction.

The user interface module is designed in such a way that at user level it accepts the query in a language understandable by expert system. To make the expert system user friendly, the user interface interacts with the user in natural language. The user interface provides as much facilities as possible such as menus, graphical interfaces etc. to make the dialog user friendly and more attractive.

2. Explanation Module: The explanation module explains the reasoning of the system to a user. It provides the user with an explanation of the reasoning process when requested. The credibility of expert system will be established only when it is able to explain "how and why" a particular conclusion is drawn. This explanation increases the belief of user in the expert system.

a) **Explanation (How):** To respond to a how query, the explanation module traces the chain of rules fired during a consolation with the user. This explanation mode can be activated once the process is over. It explains how a fact was deduced by the system and similarly how a rule was/wasn't used. The simplest way to specify this is to explain the rule which allows the deduction. For e.g.

If the system (S) will give information about the parent-child relationship to the user (U) then the followings can be possible.

S: My diagnosis is "A is the father of B"

U: How?

S: The result was obtained by the addition of following facts and rules.

Fact no 11: A is the parent of Hari.

Fact no 15: A is a male.

Fact no 110: X is father of Y:

X is parent of Y, X is male.

So, A is the father of B.

b) **Explanation (Why)?** 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. This mode of explanation is dynamic and is used when the system functions in backward chaining. If the user doesn't understand the reason behind the question. The system is capable of explaining why the question was asked. For example S: Is the following true?

A is the father of B.

U: Why?

S: I need the fact:

A is the father of B to establish the following fact “B is the son of A”.

By using the rule no. 4:

A is the father of B:

B is the son of A.

3. **Working Memory:** It is a global database of facts used by the rules.

   Knowledge Engineering: The primary people involved in building an expert system are the knowledge engineer, the domain expert and the end user. Once the knowledge engineer has obtained a general overview of the problem domain and gone through several problem solving sessions with the domain expert, he/she is ready to begin actually designing the system, selecting a way to represent the knowledge, determining the search strategy (backward or forward) and designing the user interface. After making complete designs, the knowledge engineer builds a prototype. The prototype should be able to solve problems in a small area of the domain. Once the prototype has been implemented, the knowledge engineer and domain expert test and refine its knowledge by giving it problems to solve and correcting its disadvantages.

4. **Knowledge Base:** In rule based architecture of an expert system, the knowledge base is the set of production rules. The expertise concerning the problem area is represented by productions. In rule based architecture, the condition actions pairs are represented as rules, with the premises of the rules (if part) corresponding to the condition and the conclusion (then part) corresponding to the action. Case-specific data are kept in the working memory. The core part of an expert system is the knowledge base and for this reason an expert system is also called a knowledge based system. Expert system knowledge is usually structured in the form of a tree that consists of a root frame and a number of sub frames. A simple knowledge base can have only one frame, i.e. the root frame whereas a large and complex knowledge base may be structured on the basis of multiple frames.

5. Inference Engine: The inference engine accepts user input queries and responses to questions through the I/O interface. It uses the dynamic information together with the static knowledge stored in the knowledge base. The knowledge in the knowledge base is used to derive conclusions about the current case as presented by the user's input. Inference engine is the module which finds an answer from the knowledge base. It applies the knowledge to find the solution of the problem. In general, inference engine makes inferences by deciding which rules are satisfied by facts, decides the priorities of the satisfied rules and executes the rule with the highest priority. Generally inferring process is carried out recursively in 3 stages like match, select and execute. During the match stage, the contents of working memory are compared to facts and rules contained in the knowledge base. When proper and consistent matches are found, the corresponding rules are placed in a conflict set.

KNOWLEDGE ACQUISITION

Knowledge acquisition is the gathering or collecting knowledge from various sources. It is the process of adding new knowledge to a knowledge base and refining or improving knowledge that was previously acquired. Acquisition is the process of expanding the capabilities of a system or improving its performance at some specified task. So it is the goal oriented creation and refinement of knowledge. Acquired knowledge may consist of facts, rules, concepts, procedures, heuristics, formulas, relationships, statistics or any other useful information. Source of these knowledge may be experts in the domain of interest, text books, technical papers, database reports, journals and the environments. The knowledge acquisition is a continuous process and is spread over entire lifetime. Example of knowledge acquisition is machine learning. It may be process of autonomous knowledge creation or refinements through the use of computer programs. The newly acquired knowledge should be integrated with existing knowledge in some meaningful way. The knowledge should be accurate, non-redundant, consistent and fairly complete. Knowledge acquisition supports the activities like entering the knowledge and maintaining knowledge base. The knowledge acquisition process also sets dynamic data structures for existing knowledge to refine the knowledge.

The role of knowledge engineer is also very important with respect to develop the refinements of knowledge. Knowledge engineers may be the professionals who elicit knowledge from experts. They integrate knowledge from various sources like creates and edits code, operates the various interactive tools, build the knowledge base etc.

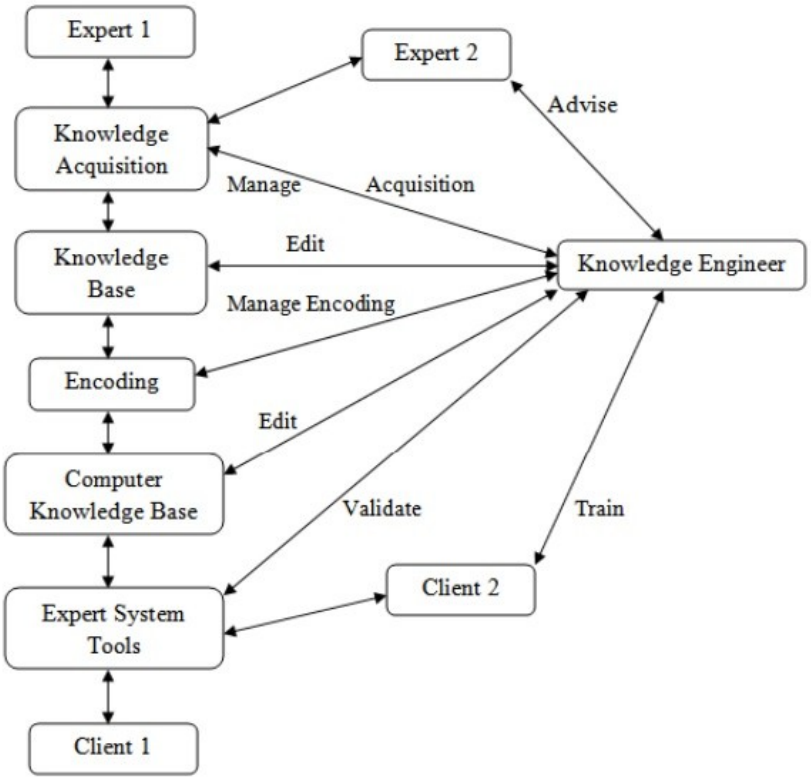


Figure Knowledge Engineer's Roles in Interactive Knowledge Acquisition

Knowledge Acquisition Techniques

Many techniques have been developed to deduce knowledge from an expert. They are termed as knowledge acquisition techniques. They are:

a) Diagram Based Techniques
b) Matrix Based Techniques
c) Hierarchy-Generation Techniques
d) Protocol Analysis Techniques
e) Protocol Generation Techniques

f) Sorting Techniques

In diagram based techniques the generation and use of concept maps, event diagrams and process maps. This technique captures the features like “why, whe n, who, how and where”. The matrix based techniques involve the construction of grids indicating such things as problems encountered against possible solutions. Hierarchical techniques are used to build hierarchical structures like trees. Protocol analysis technique is used to identify the type of knowledge like goals, decisions, relationships etc. The protocol generation techniques include various types of interviews like structured, semi-structured and unstructured.

The most common knowledge acquisition technique is face-to-face interview. Interview is a very important technique which must be planned carefully. The results of an interview must be verified and validated. Some common variations of an unstructured interview are talk through, teach through and read through. The knowledge engineer slowly learns about the problem. Then can build a representation of the knowledge. In unstructured interviews, seldom provides complete or well-organized descriptions of cognitive processes because the domains are generally complex. The experts usually find it very difficult to express some more important knowledge. Data acquired are often unrelated, exists at varying levels of complexity, and are difficult for the knowledge engineer to review, interpret and integrate. But on the other hand structured interviews are systematic goal oriented process. It forces an organized communication between the knowledge engineer and the expert. In structured interview, inter personal communication and analytical skills are important.

The current trends in Artificial Intelligence

Unless you are living under a rock, you would have come across plethora of articles convincing you that the AI revolution has come and it is here to stay. While we try to understand some of the theory behind the claims made, there would be many more articles that try to create panic among non-expert audience by conspiring doomsday theories. When there is a lack of understanding of what AI cannot do, there would be fear of what AI can do. I believe it is important that we understand the current state of the technology in the field of

AI. Recently, I had the opportunity to participate in International Conference on Machine Learning (ICML), 2017 held at Sydney, Australia. Being a prestigious ML conference, there were several amazing speakers presenting the recent advances in various subfields of ML, AI. In the rest of this article, I discuss some of the research papers that highlight a major theme in each of those subfields. The themes are based on my observations given the limited amount of time I had to attend the talks. However, for the sake of brevity of the post, I didn't mention any fundamentals of the subfields before reviewing the trends. Therefore, it needs familiarity with the concepts of these subfields in order to comprehend.

1) Reinforcement Learning and its real-world applications

While an agent is deployed in real world, it may be keen in exploring its environment but it needs to follow certain constraints in order to obey the limitations of that environment. A team from Berkeley AI Research (BAIR) has presented their work titled Constrained Policy Optimization (CPO) which induces constraints motivated by safety for policy search. It has many applications where safety can be ensured while exploration. Also, the BAIR has published an article explaining their work on CPO. If an agent/robot is purchased by a non-technical owner, she should be able to train the agent by providing feedback. McGlashan et. al has proposed Convergent Actor-Critic by Humans (COACH), an algorithm to learn from policy-dependent feedback, for training agents/robots with the feedback provided by non-technical users. They demonstrate that COACH can also learn multiple behaviours on a physical robot with noisy images as well.

In order to perform any human activity like cooking, household chores, etc., a RL agent need to execute long sequence of instructions and generalize for new unseen subtasks. Sometimes, there would be other unexpected instructions like low battery, etc., which needs a deviation to be able to finish the rest of the subtasks. To achieve these goals, Oh et. al had proposed a generalised approach which takes sequence of tasks in natural language and executes the subtasks mostly sequential. They have tackled the problem in two steps: 1) Learning the skills to perform sub tasks and an analogy based generalisation framework. 2) A meta-controller to determine the order of execution of subtasks. Unlike the existing work, their architecture generalises well and also handles unexpected subtasks. For completing a multiple set of tasks, we need a policy that can understand the sub tasks and still finish the

tasks optimising for the overall reward. Often, the agent doesn't receive an immediate reward for completing the sub task. Andreas et. al proposed a framework for learning deep sub policies in a multi task setting. The algorithm is guided only by abstract sketches of high-level behaviour.

2) Deep Learning Optimisation

In order to regularise deep neural networks, several methods like batch normalisation, whitening neural networks (WNN) are used. To apply whitening, the computational overhead of building covariance matrix and solving SVD plays a bottleneck. The work proposed by Ping Luo attempts to overcome the limitations of WNN with a new method termed Generalised Whitening Neural Networks (GWNN) which reduces computational overhead with compact representations. The limitations over hardware for implementing higher dimensional tensor kernels for ConvNets is studied by Budden et. al. They had proposed a Winograd style faster computation for higher dimensions optimised for CPUs. They have benchmarked their algorithm against popular frameworks like Caffe, Tensorflow that support the AVX and Intel MKL optimised libraries and concluded an interesting insight that the current CPU limitations are largely due to software rather than hardware.

Extending the class of faster computations, like FFT, Winograd, Cho and Brand suggested a Memory-efficient Computation (MEC) which lowers memory requirement and improves the convolution process. MEC takes rolling subsets of columns and expands them into rows to form a smaller matrix. This process is repeated along with Kernel matrix multiplication to produce efficient computation. With the increase in the number of feature maps the redundancy increases leading to inefficient memory usage. Wang et. al proposed a method called RedCNN which attempts to reduce the dimensionality of feature maps by preserving the intrinsic information and also reducing the correlation between feature maps. They used circulant matrix for projection that gives high training speed and mapping speed. Correlation between gradients decay slowly with depth in the network resulting in gradients appearing as white noise. These shattering gradients are predominantly observed in feed forward networks, however the skip-connection networks are resistant. The authors proposed Looks Linear (LL) initialisation which resolves shattering gradients in feed forward networks without adding any skip connections.

3) Deep Learning Applications

Identifying sleep patterns will help diagnose sleep disorders and thereby better healthcare can be provided. However, the existing approaches for identifying sleep patterns involve using a lot of sensors attached to patient's body and it is usually conducted in a hospital or a lab. The experimental setting itself would make the patient experience sleep difficulty rendering the measurements unreliable. A team from MIT has conducted research on using wireless radio frequency (RF) signals in identifying sleep patterns without having any sensors on patient's body. They used a CNN-RNN combination to identify patterns for sleep stage prediction. However, the RF signals suffer from noise reflected from any nearby sources in the environment. Hence, they added an adversarial training that would discard any extraneous information specific to any individual but retain the useful information required to predict the sleep stage. They had achieved significantly better results (~80%) than the existing state-of-the-art (~64%) which used hand crafted signal features. The team from Baidu has presented their work on Deep Voice, an end-to-end neural speech synthesis. They had detailed the five major building blocks that includes phoneme conversion to audio synthesis using a variant of Wavenet. As their entire architecture is powered by neural network, their system is more flexible than existing text-to-speech systems.

4) Meta Learning

Model Agnostic Meta Learning (MAML) proposed by Finn et. al, creates a meta-learned model with parameters learned from random sampling over a distribution of tasks. This model can be quickly adapted to new tasks using a few training samples and iterations, which is commonly referred as few-shot learning. The authors also demonstrated the application of MAML over classification, regression and reinforcement learning tasks. An interesting paper on learning the network structure and weights is proposed by Cortes et. al. The proposed method, called AdaNet, learns the network architecture with incremental addition of depth to the network . The new network's k^th layer is connected to existing network's k^th and k-1^th layers. The network architecture is selected by comparing their performances over an empirical loss function with regularisation parameter. Wichrowska et. al introduced a learned

gradient descent optimiser that can generalise to new tasks with reduced memory and computational requirements. They used a hierarchical RNN architecture in defining the optimiser and it outperformed RMSprop/Adam on MNIST dataset.

5) Sequential Modeling

Segmental structure is a natural pattern in many sequences like phrases in human language or group of letters in identifying phonotactic rules. Wang et. al has proposed a sequence modelling approach via segmentation. They had learned the segmental structure using LSTM and the space of possible segments are searched by keeping a limit on search space and further exploring the structure of segments. The popular implementation from Facebook AI Research (FAIR) of using Convolutions for Sequence to Sequence learning has gathered much attention at ICML'17. They created hierarchical structures using multi layer convolutions thereby replicating the long-range dependencies captured in traditional LSTM based architectures. They also used gated linear units, residual connections and attention in every decoder layer.

The temporal evolution of word embedding were studied by Bamler et. al in their paper titled "Dynamic Word Embeddings". In their approach, they extended the skip-gram to probabilistic dynamic skip grams to model sequential text data with latent time series. The key contribution of their approach is to use Kalman filter as a prior for the latent embeddings. This allowed them to share information across all times while the embeddings were allowed to drift.

6) Machine Learning Optimization

A team from Microsoft research India has come up with powerful tree based models that can help run Machine Learning in resource constraint devices like IoT with as little as 2 KB RAM. For classification problems, often Gradient Boosted Decision Trees (GBDT) performs relatively well. However, when the output space of multilabel classification becomes high dimensional and sparse, the GBDT algorithms suffers from memory issues and long running times. In order to have better prediction time and reduced model size Si et. al proposed GBDT-Sparse algorithm to handle the high dimensional sparse data.

7) Generative Models Applications

The team from Google Brain has presented a paper on audio synthesis using Wavenet auto encoders. Their main contribution being the Wavenet auto encoder architecture that includes a Temporal Encoder built over dilated convolutions that encodes sequence of hidden codes with separate dimension for time and channel. Also, they introduced NSynth dataset that contains approx 300k annotated musical notes from approx 1k instruments. Modeling sequential data like user browsing history has large action space with many actions having similar intent or topic. Recurrent neural networks like LSTM would need many parameters to model such data and makes the model highly uninterpretable. Whereas, models like LDA would model such sequential data and are interpretable but the performance is not better than LSTMs. To overcome such limitations, Zaheer et. al proposed a Latent LSTM Allocation (LLA) for user modeling combining hierarchical bayesian models with LSTMs.

The image compression algorithm proposed by Rippel and Bourdev used GANs instead of auto-encoders. The proposed solution included pyramidal decomposition encoder that extracts image features at different scales. The extracted features are decomposed into equal-sized bins using quantisation, bitplane decomposition, arithmetic coding and code length regularisation. This is followed by realistic reconstruction via adversarial training.

8) Natural Language Generation Architectures

In order to overcome the limitations of discriminative models for natural language text generation, Wen et. al proposed a Latent Intention Dialogue Model for learning the intention using latent variable and then composing appropriate machine responses. The key idea behind this paper is the representation of latent intention distribution as an intrinsic policy that reflects human decision-making and it is learned using policy gradient-based reinforcement learning. An alternative approach to the natural language generation using latent semantic structure was proposed by Hu et. al. They used VAEs to generate text samples conditioned on a latent attribute codes. The attribute codes are learned using

individual discriminator for each code that measures the match between generated samples and the desired attributes using softmax approximation.

9) Efficient Online Learning

For the online multi-class bandit algorithms, the previous work of Banditron, while computationally efficient, achieves only O(T^2/3) expected regret. This is suboptimal as the Exp4 algorithm achieves O(T^1/2) regret for the 0–1 loss. Beygelzimer et. al had proposed an efficient online bandit multi class learning with O(T^1/2) regret. The evaluation of contextual multi-armed bandits is a tough problem as the online evaluation is too costly to evaluate different policies whereas off policy evaluation methods suffer from variance in the estimations. While there exists methods like Inverse Propensity Score (IPS) which gives good estimations on the MSE, they don't consider the context information while choosing actions. The authors Wang et. al proposed an algorithm SWITCH which effectively uses the Reward model and IPS resulting in variance reduction compared to prior work.

10) Graph based algorithms

Many existing methods for generating knowledge graphs from data considers the graph to be a static snapshot. In the work published by Trivedi et. al, they had demonstrated that the knowledge graphs evolve temporally and they had developed a multidimensional point process to model the evolving knowledge graph. Identifying the transition probabilities from only the node visit counts can help in understanding the navigation behaviour of users. Maystre et. al has proposed Choice Rank, an iterative algorithm that can learn edge transition probabilities from observing only node-level traffic.