

Artificial Intelligence

ARTIFICIAL INTELLIGENCE & EXPERT SYSTEM

UNIT-1 (page 2 to page 21)

Basic Problem solving methods: Production systems-state space search, control strategies, Heuristic search, forward and backward reasoning, Hill Climbing techniques, Breadth first search, Depth first search, Best First search, Staged search.

UNIT-2 (page 22 to page 31)

Knowledge Representation: Predicate logic, Resolution question answering, non monotonic reasoning, statistical and probabilistic reasoning, semantic nets, conceptual dependency, frames and scripts

UNIT-3 (page 32 to page 45)

AI languages: Important characteristics of AI language- PROLOG, LISP

UNIT-4 (page 46 to page 50)

Introduction to Expert System, Structure of an Expert system interaction with expert, Design of an expert system.

UNIT-5 (page 51 to page 61)

Neural Network: Basic structure of neuron, perception, feed forward and back propagation, Hopfield network.

WWW.LRsir.net

UNIT-1

Basic Problem solving methods:

Production systems-state space search, Control strategies

Searching Technique:

Breadth first search

Depth first search

Heuristic search

Best First search

Hill Climbing techniques

Forward and backward reasoning

Staged search.

Introduction of Artificial Intelligence:

AI is the branch of computer science. It needs following two things:

AI कम्प्यूटर साइन्स की एक ब्रांच है। AI में निम्न दो चीजों की आवश्यकता होती है।

- 1) Thoughts and idea
- 2) Hardware and software

Definition of AI:

Artificial Intelligence is the study of human intelligence and actions replicated artificially, such that the resultant bears to its design can think and act like humans.

वास्तविक रूप से Artificial Intelligence मनुष्य की intelligence एवं उसके द्वारा किए जाने वाले कार्यों का एक अध्ययन है और इसमें सभी अध्ययनों को artificial (कृत्रिम) ढंग से दोहराने का प्रयास किया जाता है। परिणामतः एक ऐसी डिज़ाइन तैयार की जाती है जो मनुष्य के समान ही सोच विचार कर सके एवं कार्य कर सके।

Components of AI: (AI के घटक)

AI requires an understanding of related terms such as-

AI को समझने के लिए इससे संबन्धित term को जानना आवश्यक है, जैसे-

- Intelligence + Knowledge Representation
 - Various search method (विभिन्न सर्च की विधियाँ) like BFS, DFS, Heuristics
 - Reasoning and Thought
 - Cognition(ज्ञान या बोध) and Learning
 - AI languages like LISP, PROLOG
 - AI Hardware and software
 - Computer terminologies.
-

Artificial Intelligence

History of AI(Early work):

- Early 1936: **Alan Turing**, sometimes regarded as the father of AI. He had demonstrated a simple computer processor (also called Turing machine) that manipulated symbols as well as numbers.
- 1936 के पहले: Alan Turing को AI के फादर के नाम से जाना जाता है। इन्होंने एक साधारण से कम्प्यूटर प्रॉसेसर का demonstration दिया था जिसका कार्य symbols को नंबर्स में भी manipulate करना था।
- Mid 1950 is the official birth of AI.
1950 के मध्य AI का official बर्थ माना जाता है।
- Japanese were first announced fifth generation computer in October 1981 that can converse a natural language, understand speech and visual scenes, learn and refine their knowledge, make decisions and exhibit other human traits.
ओक्टोबर 1981 में जापान ने सबसे पहला 5th जेनेरेशन कम्प्यूटर की घोषणा की जो एक नैचुरल भाषा को अन्य भाषा में परिवर्तित करने, किसी आवाज और सीन को समझने, सीखने, नॉलेज को परिष्कृत करने एवं किसी मनुष्य के समान ही निर्णय लेने समर्थ रही है।

AI and related fields:

AI is generally associated with *Computer Science*, but it has many important links with other fields such as *Math*, *Psychology*, *Cognition*, *Biology* and *Philosophy*, among many others. Our ability is combine knowledge from all these fields for creating AI systems.

AI को समान्यतः कम्प्यूटर साइन्स के साथ जोड़कर देखा जाता है, किन्तु AI का संबंध ओर भी अन्य महत्व पूर्ण फील्ड के साथ है जैसे गणित, साइकॉलजी, cognition साइन्स, biology एवं philosophy ओर भी कई अन्य फील्ड जिसकी जनहा जरूरत बन सके। यह AI सिस्टम को बनाने वाले की योग्यता पर निर्भर करता है की वह किस पार्कर इन सभी फील्ड का ज्ञान एआई सिस्टम में शामिल कर सके।

Application of AI:

- Game Playing
- Speech Recognition
- Understanding Natural Language
- Expert Systems

Basic AI Problems:

There are number of Basic AI problems. Some of them are:

AI मे कई सारी बेसिक प्रॉब्लेम्स आती हे। इनमे से कुछ निम्न हे:

Water Jug Problem, Eight puzzle problem, Traveling Salesman problem etc.

To solve above type of problems we need problem solving methods like Production systems. It involves state space representation, control strategies and a number of searching technique.

उपरोक्त प्रॉब्लेम्स को सॉल्वे करने के लिए एक प्रोब्लेम सोल्विंग मेथड की आवश्यकता होगी जैसे प्रॉडक्शन सिस्टम। इसमे स्टेट स्पेस representation, कंट्रोल स्ट्रैटेजी, विभिन्न सर्च टैक्नीक आदि को शामिल किया जाता है।

General Problem Solving Method (Production System):

Production system is good way to describe system. It involves all the operation that can be performed in a search space for a solution to a problem. It is useful to structure of given problem.

किसी भी सिस्टम को समझने के लिए, प्रॉडक्शन सिस्टम सबसे सर्वश्रेष्ठ तरीका होता है। इसमे उन सभी ऑपरेशन को शामिल किया जाता है जो प्रोब्लेम का हल करने के लिए सर्च स्पेस मे शामिल किये जाना आवश्यक है। किसी भी दी गई प्रोब्लेम को सुव्यवस्थित तरीके से हल करने के लिए उसका स्ट्रक्चर बनाना बहुत उपयोगी सिद्ध होता है।

A production system consist of-

एक प्रॉडक्शन सिस्टम निम्न चीजो से मिलकर बनता है।

1. State Space Search / State Space Representation
2. Control Strategy

State space search/ State space representation:

To solve any given problem we need many things like- problem description, initial state, goal state, set of applicable rules(production rule), various searching methods etc. Then a place or space where all necessary tasks are performed called state space representation.

किसी भी प्रोब्लेम को हल करने के लिए हमे कई चीजों की आवश्यकता होती है, जैसे प्रोब्लेम का वर्णन, उसकी initial state, Goal state, production rule अर्थात लागू किए जा सकने वाले रूल का एक set, विभिन्न प्रकार की searching टैक्नीक आदि। जब इन सभी कार्यों को एक ही जगह या एक ही स्पेस मे सम्पन्न किया जाता है तब ऐसे स्पेस को ही state space representation या state space search कहा जाता है।

For a give problem state space search includes:

किसी दी गई प्रोब्लेम के लिए स्टेट स्पेस सर्च मे निम्न चीजों को शामिल करते हे:

1. State Space(Problem Description)
2. Initial State(Start)
3. Goal State(Final)
4. Production Rule(Set of Rule)

Artificial Intelligence

- 1) **State Space:** It define a state space for a given problem that contains all the possible configurations of the relevant objects.
सबसे पहले प्रोब्लेम की स्टेट स्पेस को परिभाषित करते है जिसमे संबन्धित उद्देश्य प्राप्त के लिए प्रोब्लेम को पूर्ण रूप से एवं व्यवस्थित परिभाषित किया जाता है ।
- 2) **Initial State:** Specify one or more states from which the problem-solving process may start. These states are called the initial state.
प्रोब्लेम की एक या अधिक ऐसी स्टेट को परिभाषित करना जंहा से प्रोब्लेम को हल करने की शुरुआत करनी हे। ऐसी स्टेट को initial स्टेट कहा जाता हे।
- 3) **Goal State:** Specify one or more states that would be acceptable as solutions to the problem. These states are called goal states.
प्रोब्लेम की एक या अधिक परिभाषित स्टेट जो प्रोब्लेम के हल को स्वीकार करने योग्य हो, तब उन्हें गोल स्टेट्स कहा जाता हे।
- 4) **Set of rules:** Specify a set of rules that describe the actions(operators) used to get next state of any current state.
यह विभिन्न रूल्स के समूह को परिभाषित करती हे जिनका कार्य next state के लिए करंट स्टेट्स, पर किसी एक्शन या ऑपरेशन का वर्णन करना होता हे।

The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found. Thus the process of search is fundamental to the problem-solving process.

अतः स्टेट स्पेस मे प्रोब्लेम को हल करने के लिए उपयुक्त रूल्स के कॉम्बिनेशन को किसी एक खास कंट्रोल स्ट्रेटजी(योजना) के द्वारा प्रोब्लेम स्पेस (स्टेट स्पेस) मे तब तक मुव किया जाता हे जब तक की initial स्टेट से गोल स्टेट के बीच एक पाथ प्राप्त नहीं हो जाता। अतः सर्च करने की प्रोसेस ही- प्रोब्लेम को साँल्वे करने आधारभूत तरीका हे।

Control Strategies:

To solve any problem, initially we have completely ignored the question of how to decide which rule to apply next during the process of searching. There is not a fixed control strategy for the solution of problem. It is because there may have more than one rule which have its left side match the current state.

किसी प्रोब्लेम को हल करने के लिए उसकी searching के वक्त, सबसे पहले दिमाग से एक प्रश्न निकाल देना होता हे कि आगे बडने के लिए किस रूल का चयन किया जाए। प्रोब्लेम को हल करने कि कोई निश्चित strategy (योजना) नहीं होती हे। यह इसलिए क्योंकि एक से भी अधिक ऐसे रूल होते हे जिनकी लेफ्ट साइड, करंट स्टेट से मैच होगी।

Even without a great deal of thought, it is clear that how such decision are made will have crucial impact on how quickly and even wheatear, a problem is finally solved.

यद्यपि कई बार ऐसा भी होता हे कि, बहुत अधिक विचार किए बिना ही स्थिति स्पस्ट हो जाता हे कि, ऐसा कोनसा डिसिशन बनाया जाए जो प्रोब्लेम को पूर्णतः हल करने के लिए निर्णायक, प्रभावी रहेगा और वह चाहे वह जो भी हो।

Mostly following two steps are used to set good control strategies that can be applicable to solve a problem:

प्रोब्लेम को हल करने के उसकी गुड कंट्रोल स्ट्रेटजी लिए लिए निम्न दो स्टेप का उपयोग सामान्यतः करते हे :

Artificial Intelligence

- 1) Cause Motion
- 2) Solve Systematic

Motion based:

The first requirement of a good control strategy is that it causes motion.

एक गुड कंट्रोल स्ट्रेटजी कि सबसे पहली अनिवार्यता यह हे कि, वह प्रोब्लेम कि गति(motion) को लगातार प्रभावित करती रहे।

For example: in a water jug problem, suppose we implemented the simple control strategy of starting each time at the top of the list of rules and choosing the first applicable one. If we did that, we would never solve the problem. We would continue indefinitely filling the 4-gallon jug with water.

Thus control strategy that do not cause motion will never lead to a solution.

उदाहरण के लिए: वॉटर-जग प्रोब्लेम मे, जब भी किसी रूल का चयन किया जाता हे तब उसकी शुरुआत रूल्स कि लिस्ट मे सबसे टॉप से कि जाती हे तथा जो उपयुक्त होता हे उसका चयन कर लिया जाता हे। यदि हम ऐसा करते हे तब प्रोब्लेम के हल तक कभी नहीं पहुँच सकते। हम लगातार 4-gallon वॉटर जग को वॉटर से अनिच्छित समय तक भरते ही रहेंगे।

अतः जो कंट्रोल स्ट्रेटजी गति(Motion) को प्रभावित नहीं करती उससे हल कभी प्राप्त नहीं होने वाला।

Systematically:(व्यवस्थित)

The second requirement of a good control strategy is that it be systematic.

एक गुड कंट्रोल स्ट्रेटजी कि दूसरी अनिवार्यता यह हे कि, यह पूर्णतः व्यवस्थित(systematic) होना चाहिए।

For example: In water jug problem, choosing any applicable steps, but it may be possible that one step is repeated more times. It may be happened when choosing control strategy is not systematic.

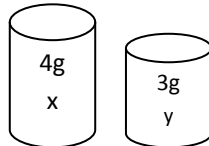
उदाहरण के लिए: वॉटर-जग प्रोब्लेम मे किसी भी उपयुक्त स्टेप को चुना तो जा सकता हे, किन्तु यह भी संभव हे कि एक ही स्टेप कई बार रिपीट होती रहे। यह तब होता हे जब बनाई जाने वाली कंट्रोल स्ट्रेटजी **systematic**(व्यवस्थित) नहीं हे।

AI Problems and Solution Strategy (Water-Jug and 8-Puzzles)

Water-Jug Problem:

We have given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring marker on it. There is a pump that can be used to fill the jugs with water. How can we get exactly 2 gallons of water into the 4 gallon jug?

हमें दो जुग दिये गए हैं- एक 4-गैलन का और एक 3 गैलन का। दोनों ही में किसी प्रकार का कोई मापक सूचक अंकित नहीं है। एक पंप है जिसका उपयोग जुग को पानी से भरने में कर सकते हैं। किस प्रकार हम 4-गैलन के जुग में 2 गैलन पानी प्राप्त कर सकते हैं।



State Space Representation (Production System of Water-Jug Problem)

1) The state space:

The state space for this problem can be described as the set of ordered pairs of integers (x,y) , such that-

वॉटर जुग प्रॉब्लेम का वर्णन करने के लिए, (x,y) integers के कई सारे क्रमागत pairs को स्टेट स्पेस में निम्न प्रकार से परिभाषित कर सकते हैं।

$$x=0,1,2,3,4$$

$$y=0,1,2,3$$

$x \rightarrow$ represents the number of gallons of water in the 4-gallons jug.

$y \rightarrow$ represents the number of gallons of water in the 3-gallons jug.

2) Initial State: (0,0)

3) Goal State: (2,n) any value of water in 3-gallon jug.

4) Production Rules:

Following assumptions can be applied to solve this problem.

इस प्रॉब्लेम को हल करने के लिए निम्न अनुमान लगाया जा सकता है।

a) Jug can be filled with pump.

जुग को केवल पंप से ही भरा जा सकता है।

b) We can pour water out of the jug onto the ground.

हम जुग के वॉटर को ग्राउंड पर पूरा फेंक सकते हैं।

c) No other measuring devices are available.

किसी भी प्रकार का कोई मापक यंत्र नहीं है।

On the basis of above assumptions we can create a number of production rules in the form of:

$$(x,y) \rightarrow (x',y')$$

Left side (x,y) of any rule can be matched against with current state. Whereas its right side (x',y') gives new state to obtain a result.

उपरोक्त assumption के आधार पर हम $(x,y) \rightarrow (x',y')$ के रूप में कई सारे प्रॉडक्शन रूल्स को बना सकते हैं। किसी भी रूल के लेफ्ट साइड (x,y) को प्रॉब्लेम की करंट स्टेट से मैच किया जा सकता है। जबकि परिणाम तक पहुंचने के लिए, राइट साइड (x',y') प्रॉब्लेम की नैक्सट स्टेट प्रदान करता है।

Artificial Intelligence

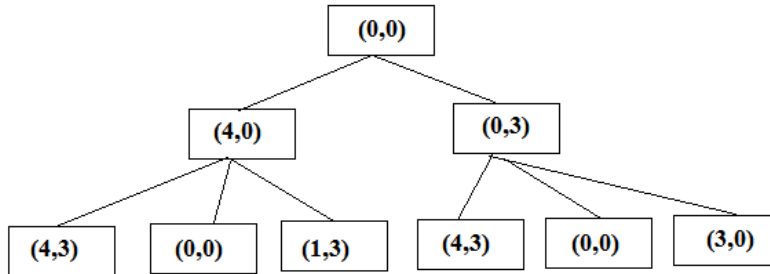
In this way maximum production rules can be generated in following way.
इस प्रकार से कई सारे प्रॉडक्शन रूल को generate किया जा सकता है।

Rule-1	$(x, y) \rightarrow (4, y)$ If $x < 4$	Fill the 4-gallon jug 4-गेलन जग को पूरा भरना
Rule-2	$(x, y) \rightarrow (x, 3)$ If $y < 3$	Fill the 3-gallon jug 3-गेलन जग को पूरा भरना
Rule-3	$(x, y) \rightarrow (x-d, y)$ If $x > 0$	Pour some water out of 4-gallon jug 4-गेलन जग में भरा कुछ वॉटर फेंक देते हैं।
Rule-4	$(x, y) \rightarrow (x, y-d)$ If $y > 0$	Pour some water out of 3-gallon jug 3-गेलन जग में भरा कुछ वॉटर फेंक देते हैं।
Rule-5	$(x, y) \rightarrow (0, y)$ If $x > 0$	Empty the 4-gallon on the ground 4-गेलन जग का पूरा वॉटर फेंक देते हैं।
Rule-6	$(x, y) \rightarrow (x, 0)$ If $y > 0$	Empty the 3-gallon on the ground 3-गेलन जग का पूरा वॉटर फेंक देते हैं।
Rule-7	$(x, y) \rightarrow (4, y-(4-x))$ If $x+y \geq 4$ & $y > 0$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full 4-गेलन जग को 3 गेलन जग के वॉटर से पूरा भर देते हैं।
Rule-8	$(x, y) \rightarrow (x-(3-y), 3)$ If $x+y \geq 3$ & $x > 0$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full 3-गेलन जग को 4-गेलन जग के वॉटर से पूरा भर देते हैं।
Rule-9	$(x, y) \rightarrow (x+y, 0)$ If $x+y \leq 4$ & $y > 0$	Pour all the water From the 3-gallon jug Into the 4-gallon jug 4-गेलन जग में 3-गेलन का पूरा वॉटर डाल देते हैं।
Rule-10	$(x, y) \rightarrow (0, x+y)$ If $x+y \leq 3$ & $x > 0$	Pour all the water From the 4-gallon jug Into the 3-gallon jug 3-गेलन जग में 4-गेलन का पूरा वॉटर डाल देते हैं।
Rule-11	$(0, 2) \rightarrow (2, 0)$	Pour 2 gallons water From the 3-gallon jug Into the 4-gallon jug 2- गेलन वॉटर को 3-गेलन वॉटर से 4-गेलन वॉटर में डाल देते हैं।
Rule-12	$(2, y) \rightarrow (0, y)$	Empty the 2 gallons water of 4-gallon jug on the ground. 4-गेलन जग में से 2-गेलन का वॉटर ग्राउंड पर फेंक देते हैं।

Artificial Intelligence

Control Strategies for Water-Jug problem :

We use any one searching technique like BFS, DFS etc to solve this problem as—
प्रोब्लेम को सर्च करने के लिए किसी भी सेयरचिंग टैक्नीक का उपयोग कर सकते हे जैसे-BFS, DFS आदि-



and we get one solution in following order.

एवं इस प्रकार से हमे निम्न क्रम मे हल प्राप्त होता हे।

Gallons in the 4-gallon jug(x)	Gallons in the 3-gallon jug(y)	Rule Applied (x,y)→(x',y')
0	0	
0	3	Rule-2
3	0	Rule-9
3	3	Rule-2
4	2	Rule-7
0	2	Rule-5 or 12
2	0	Rule-9 or 11

Problem Characteristics:

- 1) Non decomposable.
- 2) Recoverable or undone
- 3) Solution is universal predictable.
- 4) Solution is absolute.
- 5) Solution has a path.
- 6) A little amount of knowledge is required.
- 7) Does not need human interaction.

Artificial Intelligence

Ex: 8-Puzzle Problem

Problem Description: A frame that can contain 9 tiles of equal size (3-3-3 in each row), but it contains 8 square tiles, each labeled 1 to 8. 9th one is uncovered. Nearest tiles can be moved to the uncovered place. In this way, blank space can be moved towards up, down, left, and right. Blank space can be set in any position, and we move squares to find the given tiles' position.

प्रोब्लेम का वर्णन: एक ऐसी फ्रेम लेते हैं जिसमें एक ही आकार की 9 टाइल्स (प्रत्येक रो में 3-3-3) को रखा जा सके, किन्तु इसमें केवल 8 ही टाइल्स को रखा जाता है। प्रत्येक पर लेबल 1 से 8 अंकित है। 9वां ब्लैंक स्पेस है। इस ब्लैंक स्पेस में आसपास की टाइल्स मुव हो सकती हैं जिससे वह ब्लैंक स्पेस up, down, left या right में मुव हो जाता है। ब्लैंक स्पेस किसी भी पोजिशन में हो सकता है और वही से दी गई टाइल की पोजिशन मालूम करने के लिए टाइल्स को मुव करना शुरू कर देते हैं।

State Space: 8-puzzle look like- 8 पज़ल निम्न प्रकार का हो सकता है-

1	2	3
4	5	6
7	8	

— Tiles

— Blank space

Here tile number 6 or 8 can move towards to blank space.

यहाँ tile 6 या 7 को ब्लैंक स्पेस की ओर मुव कर सकते हैं।

Initial State and Goal State:

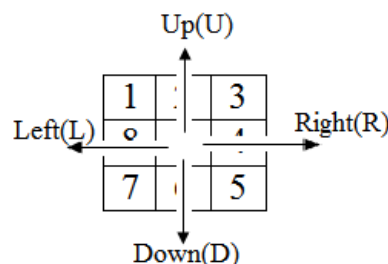
In this problem, user can decide initial state and goal state. Every time state of this problem can be altered, like:

इस प्रोब्लेम में यूजर स्वयं इनिशियल एवं गोल स्टेट डिजाइन कर सकता है। प्रत्येक बार प्रोब्लेम की स्टेट कुछ भी हो सकती है। जैसे:

Initial State	Goal State																		
<table border="1" style="display: inline-table;"><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>1</td><td>6</td><td>4</td></tr><tr><td>7</td><td> </td><td>5</td></tr></table>	2	8	3	1	6	4	7		5	<table border="1" style="display: inline-table;"><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>8</td><td> </td><td>4</td></tr><tr><td>7</td><td>6</td><td>5</td></tr></table>	1	2	3	8		4	7	6	5
2	8	3																	
1	6	4																	
7		5																	
1	2	3																	
8		4																	
7	6	5																	

Set of rules:

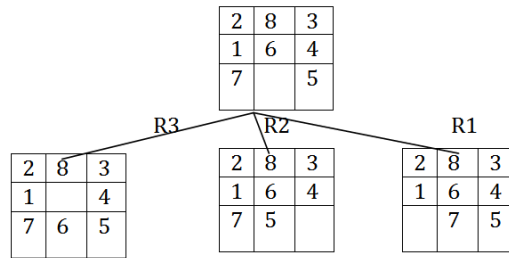
- R1 → Move blank square left
- R2 → Move blank square right
- R3 → Move blank square up
- R4 → Move blank square down



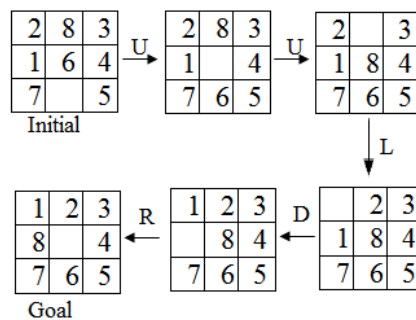
Artificial Intelligence

Control Strategies:

We use any one searching technique like BFS, DFS etc to solve this problem as—
 प्रोब्लेम को सर्च करने के लिए किसी भी सेयर्चिंग टैक्नीक का उपयोग कर सकते है जैसे-BFS, DFS आदि-



In this way path of solution can be drawn as- इस प्रकार सोल्यूशंस का पाथ निम्न प्रकार से ड्रॉ होता है।



Problem Characteristics:

- 1) Non decomposable.
- 2) Recoverable or undone
- 3) Solution is universal predictable.
- 4) Solution is absolute.
- 5) Solution has a path.
- 6) A little amount of knowledge is required.
- 7) Does not need human interaction.

Searching Technique:

Problems are typically defined in terms of states and solutions correspond to goal states. Solving a problem then amounts of searching through the different states until one or more of the goal states are found.

दी गई प्रॉब्लेम्स कई सारी states में होती है तथा उसका हल किसी एक goal स्टेट्स के रूप में परिभाषित रहती है। प्रोब्लेम को हल करते समय एक स्टेट से अन्य स्टेट पर सर्च करते हुए पहुँचते है जब तक की एक या अधिक गोल स्टेट प्राप्त नहीं हो जाती।

In this way a number of searching techniques have discovered. Each one has some benefits and limitations. Some of them are following.

इस प्रकार से बहुत सारी searching टैक्नीक विकसित हो चुकी है। प्रत्येक के अपने कुछ लाभ है एवं कामिया भी। कुछ सेयर्चिंग टैक्नीक निम्न है।

- 1) Breadth First Search
- 2) Depth First Search

- 3) Heuristic Search
- 4) Best First Search
- 5) Hill Climbing Search
- 6) Forward & Backward Reasoning

Breadth First Search: (B.F.S.)

It is one form of Blind and uninformed search because this search begins from initial state and does not use any other information to select a particular next state.

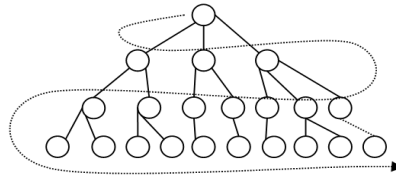
In this searching technique we explore all possible next states of current state before proceeding next level. This means all immediate children nodes of current node will be generated before generating children's children. This process will be continued until any child node does not match with goal state.

To search goal state, a tree is designed that begins with initial state, consider as root. Apply all possible production rules on initial state to generate next level of nodes.

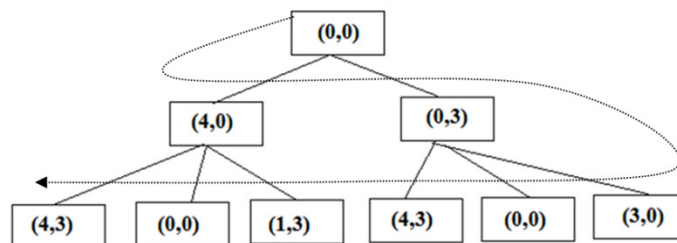
यह blind तथा uninformed सर्च का एक तरीका है, क्योंकि इस सर्च में शुरुआत इनिशियल स्टेट से होती तो है किन्तु उसकी सभी next स्टेट में से आगे बढ़ने के लिए किसे चुना जाए, इसकी जानकारी का उपयोग नहीं होता।

इस searching तकनीक में हम किसी current स्टेट की सबसे पहले सभी next स्टेट को explore करते हैं उसके बाद ही उसके next लेवल पर जाते हैं। इसका अर्थ यह है की current node की सभी चिल्ड्रेन नोड्स, उनकी भी चिल्ड्रेन नोड जनरेट होने से पहले ही generate होगी। यह प्रोसेस तब तक चलती रहती है जब तक की कोई चीलड नोड गोल नोड से मैच न हो जाए।

गोल स्टेट को सर्च करने के लिए एक ट्री को डिजाइन करते हैं जो प्रोब्लेम की इनिशियल स्टेट से शुरू होती है, जिसे रूट माना जाता है। इनिशियल स्टेट पर नैक्सट लेवल की सभी नोड्स प्राप्त करने के लिए सारे प्रॉडक्शन रूल अप्लाई करते हैं।



Example: Water Jug Problem



Advantages:

- 1) It is quite simple as compared to other.
अन्य की तुलना में यह बहुत सरल तकनीक है।
- 2) Always finding a minimal path length solution when one exist.
यदि किसी प्रोब्लेम का हल है तब यह मेथड सबसे कम length के path के हल को प्रदान करती है।

Disadvantages:

- 1) A great many nodes may need to be explored before a solution is found.
हल के आने से पहले कई सारी nodes को explore करना होता है।
- 2) Need a lot of time.(more time complexity)
बहुत समय लगता है प्रोब्लेम के हल आने में।
- 3) Need a lot of space(more space complexity)
बहुत अधिक स्पेस की आवश्यकता होती है।
- 4) Searching process remembers all unwanted nodes which is not a specific use.
इस मेथड में सभी अवांछित nodes को भी बनाकर रखना होता है जिनका कोई विशेष उपयोग नहीं है।

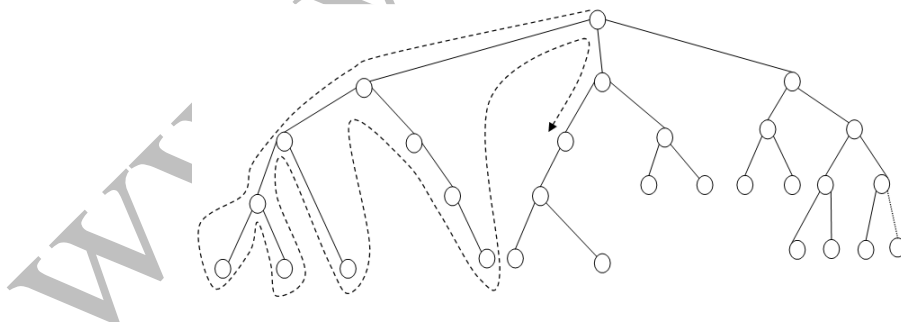
Depth First Search: (DFS)

It is also one form of Blind and uninformed search. DFS is performed by diving downward into a tree as quickly as possible. It does this by always generating a child node from the most recently expanded node, then generating that child's children, and so on until a goal node is found or some cutoff depth point d is reached.

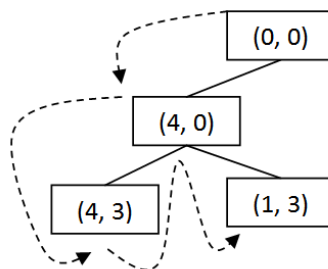
If a goal node is not found when a leaf node is reached or at the cutoff point, the program backtracks to the most recently expanded node and generates another of its children. This process continues until a goal is found or failure occurs.

यह भी एक Blind तथा uninformed सर्च तकनीक का एक रूप है। इस सर्च मेथड में प्रोब्लेम के ट्री संरचना में प्रारम्भिक स्टेट(रूट) से नीचे की ओर लगातार गहराई में जाना होता है। ऐसा करने के लिए सबसे पहले करंट नोड की किसी एक child नोड को गेनरेट करते हैं। अन्य child नोड पर ध्यान दिये बिना पहली चाइल्ड नोड को करंट नोड मानकर उसकी चाइल्ड नोड बनाते हैं, इस प्रकार यह प्रोसेस ट्री के depth में जाने लगती है, जब तक की प्रोब्लेम की गोल स्टेट नहीं आ जाती या cutoff पॉइंट तक नहीं पहुँच जाते या लीफ नोड प्राप्त नहीं हो जाती।

यदि गोल स्टेट आने के पहले ही लीफ नोड या cutoff पॉइंट आ जाता है तब उसकी पेरेंट नोड की अन्य चाइल्ड नोड को प्राप्त करते हैं। यह प्रोसेस लगातार पूरे ट्री स्ट्रक्चर में बनी रहती है जब तक प्रोब्लेम की गोल स्टेट प्राप्त नहीं हो जाती या सिस्टम काम करना बंद नहीं कर देता।



Ex: Water Jug problem



Artificial Intelligence

Advantages: The depth first search is preferred over the Breadth First Search when the search tree is known to have a plentiful number of goals.

ब्रेड्थ फर्स्ट सर्च की तुलना में डेप्थ फर्स्ट सर्च को सबसे अधिक महत्व दिया जा सकता है यदि बहुत अधिक मात्रा में गोल प्राप्त करना हो।

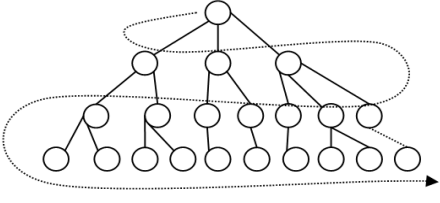
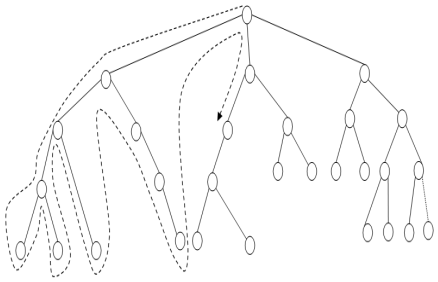
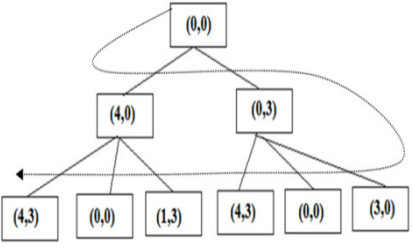
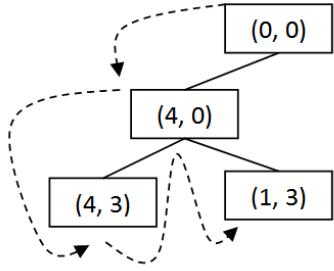
Limitation: DFS may never find a solution for a single goal state. The depth cutoff also introduces some problems. If it is set to shallow, goals may be missed, if set to deep, extra computation may be performed.

प्रोब्लेम की केवल एक ही गोल स्टेट के लिए DFS बिल्कुल भी उपायुक्त नहीं है। इस सर्च में डेप्थ cutoff से भी बहुत समस्याएं आती हैं। यदि इसे कम लेवल पर सेट करें तो गोल छूट सकता है और यदि अधिक लेवल पर सेट करें तो अधिक computation करना होगा गोल तक जाने के लिए।

Difference between Breadth First Search and Depth First Search:

S.n.	BFS	DFS
1	We place all the children at the end of the queue in any order. क्यू के अंत में सभी चिल्ड्रेन को किसी भी क्रम में रखा जाता है।	Place all children at the front of the queue. क्यू के फ्रंट में सभी चिल्ड्रेन को रखा जाता है।
2	Need large memory space. Means have large space complexity. अधिक मेमोरी स्पेस की आवश्यकता होती है। अर्थात् इसकी स्पेस complexity अधिक होती है।	Need less memory space comparatively. तुलनात्मक रूप से कम मेमोरी स्पेस की आवश्यकता होती है।
3	Space complexity= $O(b^d)$	Its has Space complexity= $O(d)$
4	Need large amount of time to reach goal node. Means have large time complexity. गोल नोड तक जाने के लिए बहुत अधिक समय लगता है। अर्थात् इसकी टाइम complexity अधिक होती है।	It may also need great amount of time to reach goal node. Means have large time complexity. गोल नोड तक जाने के लिए इसे भी अधिक समय लगता है। अर्थात् इसकी टाइम complexity अधिक होती है।
5	Time complexity= $O(b^d)$	Same Time complexity= $O(b^d)$
6	It is sure that we found goal state. हम पूरी तरह से आश्वस्त रहते हैं कि, गोल नोड प्राप्त होगी ही।	we are not sure to found goal state. हम पूरी तरह से आश्वस्त नहीं रहते हैं कि, गोल नोड प्राप्त होगी ही।
7	It has not any cutoff point d for returning to previous level node. इसमें ऐसा कोई cutoff पॉइंट नहीं होता जहाँ से पिछली नोड पर जा सके।	It has a cutoff point d for returning to previous level node. इसमें एक cutoff पॉइंट होता है जहाँ से पिछली नोड पर लौटा जाता है।
8	Suitable for a goal state solution. एक ही गोल प्राप्त करने के लिए श्रेष्ठ है।	Suitable for more goal state solution. एक से अधिक गोल प्राप्त करने के लिए श्रेष्ठ है।

Artificial Intelligence

<p>9</p>	<p>Flow diagram:</p> 	<p>Flow diagram:</p> 
<p>10</p>	<p>Example: Water Jug</p> 	<p>Example: Water</p> 

www.LRSir.net

Heuristic Search Technique:

Heuristics is one type of informed search because in the state space search, *Heuristics define the rules* for choosing branches that are most likely to lead to an acceptable solution.

हयूरिस्टिक्स एक इन्फॉर्मड प्रकार की सर्च टैक्नीक है। इसमे ऐसे रूल्स को परिभाषित करते हे जिनकी सहायता से प्रोब्लेम की उन्ही ब्रांच का चयन करते है जिस ओर सोल्यूशंस प्राप्त होने की अधिक उम्मीद हो।

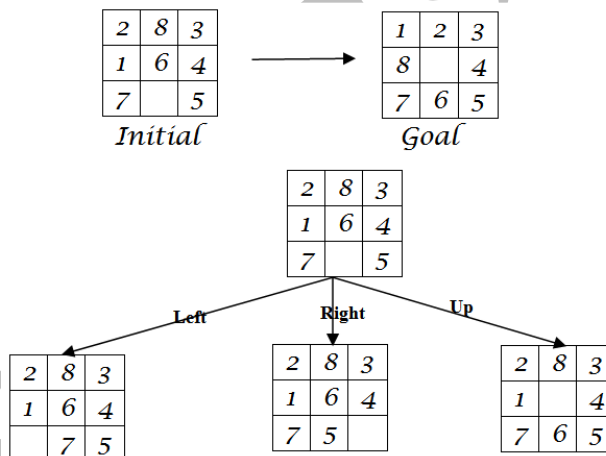
Searching Strategy of Heuristic: Heuristic is like Breadth First Search, except that this search method does not proceed uniformly outward from start node; instead, it proceeds preferentially through nodes that heuristics, problem-specific information indicates might be on the best path to a goal.

यह सर्च टैक्नीक breadth सर्च टैक्नीक के समान ही होती है, किन्तु इस सर्च विधि मे बिना किसी जानकारी के स्टार्ट नोड से आगे की ओर बड़ा नहीं जाता है। अर्थात सबसे पहले सभी नोड्स की हयूरिस्टिक ज्ञात करते है और उनमे से जो सबसे श्रेष्ठ हो केवल उसी नोड को गोल की दिशा मे बेस्ट पाथ मानकर आगे बड़ जाते है।

Heuristic Function: A function which is used to help to decide which node is best one to expand next called heuristics function.

एक ऐसा function जिसकी सहायता से उपलब्ध एक से अधिक नोड मे से next विस्तार के लिए केवल किसी एक नोड का चयन करने के लिए करते है।

Ex: 8-puzzle problem



Here which move is best? So we find Heuristic Value of each node to proceed further expand. We need a Heuristic function to evaluate Heuristic value. Like-

उपरोक्त उदाहरण मे- बेस्ट मुव कौनसा है? इसके लिए हम सबसे पहले प्रत्येक नोड की हयूरिस्टिक वैल्यू ज्ञात करते हैं। इसके लिए हमे एक हयूरिस्टिक फंक्शन की आवश्यकता होगी। निम्न प्रकार से--

Heuristic function: “Count how far away(How many tiles movement) each tile is from it correct position. Sum of this count over all tiles and expand which has less number.”

हयूरिस्टिक फंक्शन: “प्रत्येक टाइल अपनी वास्तविक स्थिति से कितनी दूर है (अर्थात कितनी टाइल को मुव करना है)उन सभी की गिनती करते हैं। सभी टाइल के गिनती का योग करते है एवं अगली नोड का विस्तार करने के लिए केवल उसी का चयन करते है जिसका योग सबसे कम आया हो।

In Above example, we find heuristic value of each expanded node to select next expanded.

Artificial Intelligence

उपरोक्त उदाहरण में, expand की गई प्रत्येक नोड की ह्यूरिस्टिक वैल्यू जात करते हैं जिससे की नैक्सट expand का चयन कर सके।

1) Left Move: Tiles are away from correct position:

Tile Number:	1	2	3	4	5	6	7	8
Count Move:	1	1	0	0	0	1	1	2
Sum of all move=	$(1+1+0+0+0+1+1+2)=6$							

2) Right Move: Tiles are away from correct position:

Tile Number:	1	2	3	4	5	6	7	8
Count Move:	1	1	0	0	1	1	0	2
Sum of all move=	$(1+1+0+0+1+1+0+2)=6$							

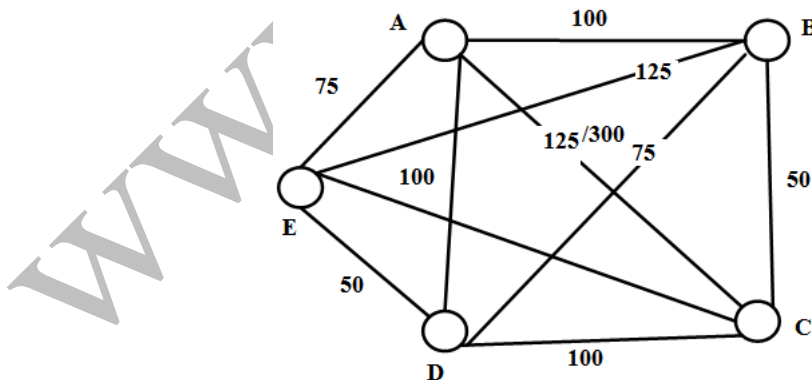
3) Up move: Tiles are away from correct position:

Tile Number:	1	2	3	4	5	6	7	8
Count Move:	1	1	0	0	0	0	0	2
Sum of all move=	$(1+1+0+0+0+0+0+2)=4$							

According to heuristic value “Up” movement is better because -

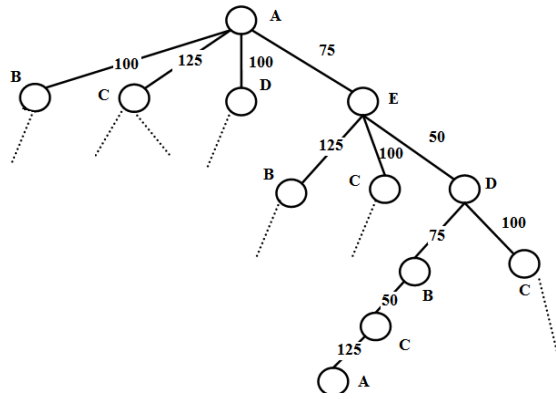
Tiles are away from correct position=4(Lowest)

Example: Travelling sales man problem.



It is too complex to be solved via direct/blind search for large number of cities(N) .
 ‘The nearest neighbor heuristic’ works well most of the time, but with some arrangement of cities, it does not find the shortest path.

Heuristic search space can be represented as-



This heuristic search work well when the distance between city C—A is 125, because total distance covered is $75(AE)+50(ED)+75(DB)+50(BC)+125(CA\text{-return})=375$.

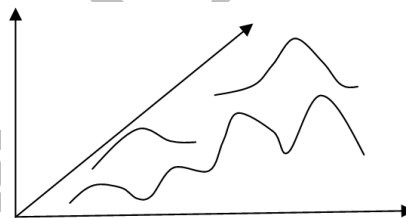
But this heuristic becomes fail if distance between CA is 300 because distance from C to A via C-B-A($50+100=150$) is less then direct path ($CA=300$). But it is not possible in this problem state that any visited city should not repeated and path always minimal when covered all cities.

On the basis of above example we say that Heuristic select more promising nodes which works successfully in many cases but its success is not guaranteed.

Hill Climbing Searching

This searching is based on hill-climbing using altimeter to measure fog pressure. Person will move towards low pressure.

यह सर्च टैक्नीक “कोहरे का दबाव मापने वाले एलिमेटर यंत्र की सहायता से पहाड़ पर चढ़ना” पर आधारित है। पर्सन केवल उसी ओर मुव करेगा जिस ओर कम दबाव हो।



So in this searching nodes are selected in similar ways for expansion. At each point in search path, a successor node that appears to lead most quickly to the top of hill (the goal) is selected for expansion.

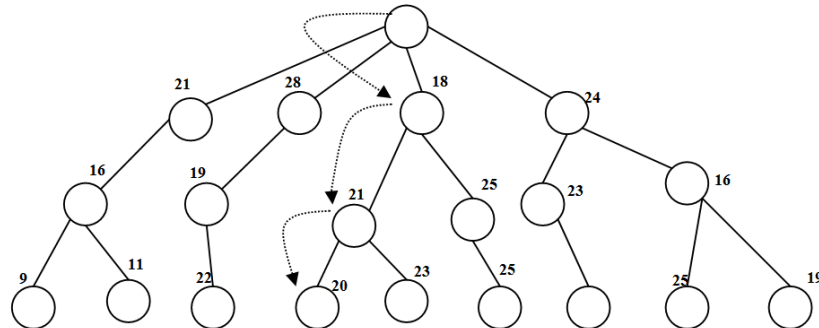
अतः इस सर्च विधि मे नोड्स को केवल इसी प्रकार से विस्तार के लिए चयन करते है। सर्च पाथ मे किसी भी पॉइंट पर केवल उसी successor नोड को विस्तार के लिए सिलैक्ट करते हे जिसकी उम्मीद hill के टॉप (गोल नोड) पर जाने की अधिक हो।

Hill climbing is informed and like depth first search technique where the most promising child is selected for expansion. When the children have been generated, alternative choices are evaluated using some types of Heuristic function. The path that appears most promising is then chosen and no further reference to the parent or other children is retained. This process continues from node to node with previously expanded nodes being discarded.

हिल क्लिंबिंग एक इन्फॉर्मड सर्च एवं depth फ़र्स्ट सर्च के समान है जन्हा expansion के लिए मोस्ट प्रोमिसींग चीलड़ नोड को सिलैक्ट करते है। जब सभी चिल्ड्रेन नोड्स generate की जा चुकी हो, तब किसी एक नोड

Artificial Intelligence

का चयन करने के लिए विभिन्न प्रकार के heuristic फंक्शन के द्वारा alternet चॉइस को ज्ञात किया जाता है। जो पाथ सबसे अधिक प्रोमिसींग लगता है उसका चयन कर पेरेंट नोड और अन्य चिल्ड्रेन नोड्स का रिफ्रेंस रखे बिना आगे बाद जाते है। यह प्रोसेस नोड से अन्य नोड तक निरंतर बनी रहती है।



Advantages:

- It is relatively simple to implement.
- Reduces number of visited nodes.
- In some situation, it give better solution in a limited time.

लाभ:

- तुलनात्मक इसे लागू करना अधिक सरल होता है।
- Visit की जा चुकी नोड को कम करता है।
- कुछ परिस्थितियों में यह सीमित समय में अच्छे परिणाम देती है।

Drawbacks: कमियाँ

1) Local Maxima:

A state that is better than all neighbors but is not better than some other state farther away. At a local maximum, all moves appear to make things worse.

वह स्टेट जो आसपास की सभी स्टेट से श्रेष्ठ हो किन्तु उसके बाद की अन्य स्टेट से श्रेष्ठ नहीं हो। लोकल मैक्सिमम पर सभी मुव बेकार साबित होने लगते हैं।

2) Plateau:

A plateau is flat area of search space in which a whole set of neighboring state has same value. It is not possible to determine the best direction in which to move by making local comparison.

सर्च स्पेस में plateau वह समतल एरिया है जिसमें सभी आसपास की स्टेट्स की एक समान वैल्यू हो। ऐसे में आगे मुव करने के लिए बेस्ट डाइरेक्शन ज्ञात करना संभव नहीं है।

3) Ridge:

It is special kind of local maxima. It is an area of search space that is higher than surrounding areas and that itself has a slop.

यह लोकल maxima का विशेष प्रकार है। सर्च स्पेस में यह वो एरिया होता है जो आसपास के एरिया से अधिक हो और वह खोद एक ढलान हो।

Best First Search:

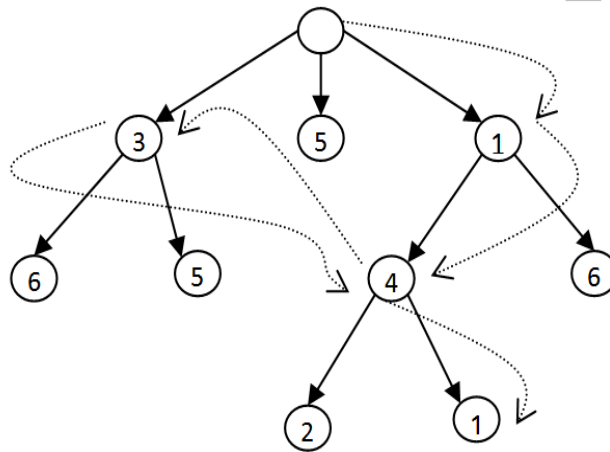
This searching technique is based on the use of heuristic to select most promising paths to the goal node. In this searching, all estimates computed for previously generated nodes and make its selection based on the best among them all.

गोल नोड को प्राप्त करने के लिए यह सर्च टैक्नीक heuristic का उपयोग करते हुये मोस्ट प्रोमिसींग पाथ का चयन करती है। इस सर्च विधि मे सभी generate किए जा चुके previous नोड्स का estimates ज्ञात करते है, एवं उन सभी मे से जो बेस्ट हे उनका चयन आगे बड़ने के लिए कर लेते है।

Thus at any point in the search process, best first moves forward from the most promising of all the nodes generated so far.

अतः बेस्ट सर्च की प्रोसैस मे किसी भी पॉइंट से केवल उसी दिशा मे आगे बढते हे जिस ओर प्रॉबलम का हाल आने की संभावना अधिक हो।

Example:



Best First Search V/S Breadth & Depth First Search

In Best First Search we expand all nodes of current node (like Breadth First Search) but select any one most promising node to goal and switch down to generate next nodes (like Depth First Search).

बेस्ट फ़र्स्ट सर्च मे breadth फ़र्स्ट सर्च के समान किसी करेंट नोड की सभी नैक्सट नोड को expand करते है किन्तु उनमे से केवल किसी एक ऐसी नोड का चयन करते हे जिसके आगे बड़ने पर गोल नोड प्राप्त होने की संभावना अधिक हो। और depth फ़र्स्ट नोडे के समान सभी नोड को छोड़कर एक ही नोडे को लेकर नीचे की ओर स्विच हो जाते है।

Forward and Backward reasoning (Bidirectional Search)

When a problem has a single goal state that is give explicitly then bidirectional search technique can be used.

जब किसी प्रोब्लेम की केवल एक ही गोल स्टेट हो जिसे पहले से बताया जा चुका हो तब bidirectional सर्च टैक्नीक का उपयोग किया जा सकता है।

In this reasoning process, searching of solution is perform using following two process-

इस रीज़निंग प्रोसैस मे, प्रोब्लेम का सोल्यूशंस निम्न दो प्रोसैस मे पूरा किया जाता है।

Artificial Intelligence

1) Forward Reasoning:

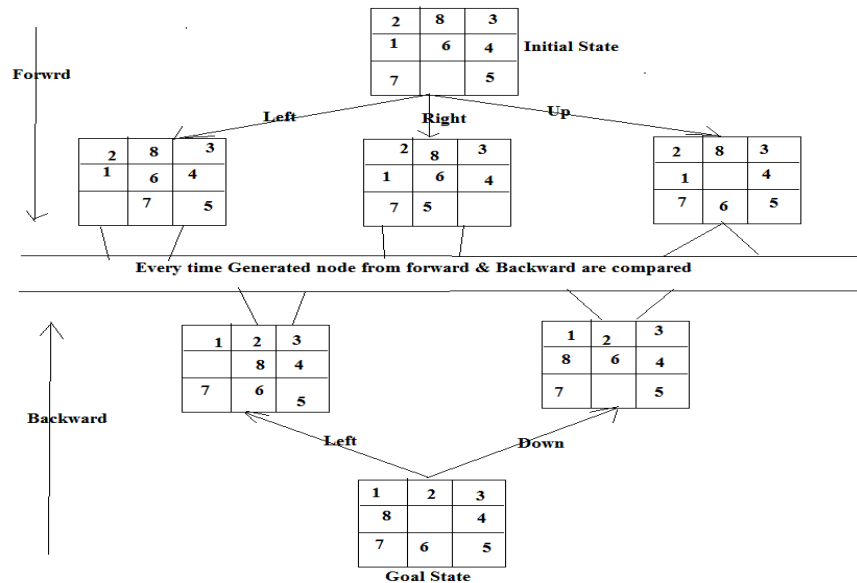
In which we apply any suitable search on initial state and moves towards goal state.

इसमें किसी भी उपयुक्त सर्च टैक्नीक को प्रारम्भिक स्टेट पर अप्लाई करते हैं और गोल स्टेट की ओर मुव करते हैं।

2) Backwards Reasoning:

In which we apply any suitable search on goal state and moves towards Initial state.

इसमें भी किसी भी उपयुक्त सर्च टैक्नीक को गोल स्टेट पर अप्लाई करते हैं और प्रारम्भिक स्टेट की ओर मुव करते हैं।



At the same time, forward and backward processes are done. When we get a common next state of both reasoning, stop node generating process and trace path from initial state to goal state which is minimal path.

एक साथ एक ही समय में फॉरवर्ड और बैकवार्ड प्रोसेस को अप्लाई किया जाता है। जब दोनों रीजनिंग के द्वारा हमें कोई कॉमन नैक्सट स्टेट मिलती है तब generating प्रोसेस को रोक देते हैं और प्रारम्भिक स्टेट से गोल स्टेट के बीच के पाथ को ट्रेस कर लेते हैं।

UNIT-2

Knowledge Representation:

- Predicate logic
- Resolution question answering
- semantic nets
- conceptual dependency
- frames
- scripts
- non monotonic reasoning
- statistical and probabilistic reasoning

Knowledge Representation:

In order to solve the complex problems encountered in AI, one generally needs a large amount of knowledge, and suitable mechanisms for representing and manipulating all that knowledge. Knowledge can take many forms. Some simple examples are:

AI में किसी भी जटिल समस्या के प्राप्त होने पर सामान्यतः अधिक मात्रा में नॉलेज की आवश्यकता होती है तथा इस नॉलेज को प्रदर्शित और क्रियान्वन में लाने के लिए किसी उपायुक्त मैकानिज़म हो। नॉलेज के बहुत सारे रूप होते हैं। इसके कुछ उदाहरण निम्न हैं।

- John has an umbrella.
- It is raining.
- An umbrella stops you getting wet when it's raining.
- An umbrella will only stop you getting wet if it is used properly.
- Umbrellas are not so useful when it is very windy.

So, we need to study, how should an AI agent store and manipulate knowledge like this? Knowledge Based System can become effective as problem solvers only when specific knowledge was brought to bear on the problems. Knowledge representation needs two different entities:- fact and Representation of facts.

अतः हमें इस बात का अध्ययन करने की आवश्यकता है की किस प्रकार AI एजेंट इस प्रकार से नॉलेज को स्टोर और मैनिपुलेट करे? नॉलेज बेस्ड सिस्टम केवल विशेष प्रकार की प्रोब्लेम को ही सोल्व करने के लिए ही प्रभावशील होते हैं। नॉलेज को व्यक्त करने के लिए फ़ैक्ट और उनका प्रदर्शन दोनों की आवश्यकता होती है।

Predicate logic v/s propositional logic:-

Propositional logic is simple to deal. We can easily represent real world facts as logical propositions written as well formed formula(wff's).

किसी भी प्रकार के व्यवहार के लिए propositional लॉजिक बहुत सरल होता है। हम वास्तविक दुनिया से संबन्धित तथ्यों को बहुत ही आसानी से लॉजिकल proposition के रूप में लिखे जाने वाले सूत्र में व्यक्त कर सकते हैं।

Ex:

It is raining -*RAINING*

It is Sunny -*SUNNY*

Artificial Intelligence

If it is raining, then it is not sunny.

$RAINING \rightarrow \neg SUNNY(\neg \text{ for not})$

Limitation: कमियाँ

Suppose we want to represent the obvious fact stated by sentence.

माना कि हम कुछ वाक्यों को इस प्रकार लिखते हैं -

- i) Scocrates is a man. $-SCORATESMAN$
- ii) Plato is a man. $-PLATOMAN$

Which would be totally separated assertion(दावा) and we would not be able to draw any conclusion about similarities between Scocrates and Plato.

इन दोनों वाक्यों में एक ही बात पर अलग अलग जोर दिया गया है किन्तु स्कोरटेस और प्लेटो के बीच किसी भी समानताओं को बता नहीं सकते।

It would be much better to represent these facts as-

इन तथ्यों को निम्न प्रकार से व्यक्त करना अधिक श्रेष्ठ है।

$-MAN(SCORATES)$

$-MAN(PLATO)$

- iii) All men are mortal $-MORTALMAN.$

This representation fails to capture the relationship between any individual being a man and that individual being a mortal. Then we write a separate statement about the mortality of every known man. In this case, we use predicate logic to represent real world facts as statement written as wff(well formed formula)

यह प्रस्तुतीकरण, किसी एक man और वह एक mortal (घातक) है, इन दोनों के बीच संबंध को हासिल करने में असमर्थ रहता है। ऐसी स्थिति में प्रत्येक man की मोर्टलिटी के बारे में अलग से कथन लिखना होगा। इस प्रकार के केस में, हम प्रेडिकेट लॉजिक का उपयोग वास्तविक दुनिया से संबन्धित तथ्यों को wff स्टेटमेंट (वेल फॉर्मड फॉर्मूला) में लिखते हैं।

Advantage of predicate logic over propositional logic:-

1. Determining the validity of a proposition in propositional logic is straight forward but it may be computationally hard.

Propositional लॉजिक में किसी proposition(MORTALMAN) की वैधता को ज्ञात करना आसान होता है किन्तु कम्प्यूटर द्वारा प्रोसेस करना बहुत कठिन होता है।

2. Predicate logic provides a good way to represent knowledge.

किसी नॉलेज को प्रेडिकेट लॉजिक द्वारा अच्छे से व्यक्त किया जा सकता है।

3. It provides a deducing new statement from old one.

प्रेडिकेट लॉजिक उपलब्ध कथनों से नए कथन ज्ञात कर सकती है।

4. Propositional logic does not possess a decision procedure.

Propositional लॉजिक में निर्णयात्मक विधि नहीं होती।

Artificial Intelligence

Characteristics of predicate logic:-

1. It uses formal application of reasoning, necessary to automate reasoning process.
प्रेडिकेट लॉजिक, मे रीज़निंग प्रोसेस के लिए रीज़निंग का औपचारिक अनुप्रयोग होता है।
2. FOPL (First Order Predicate Logic) is flexible and it uses accurate representation of knowledge and natural language.
FOPL(फ़र्स्ट ऑर्डर प्रेडिकेट लॉजिक) की नेचर लचीला होने के कारण इसका उपयोग नॉलेज और नैचुरल भाषा को अच्छे से व्यक्त करने के लिए कर सकते हैं।
3. FOPL is widely accepted by workers in the AI field.
एआई के क्षेत्र मे कर्मचारियों द्वारा FOPL का सबसे अधिक उपयोग किया जाता है।
4. It is used in programming design and literature.
इसका उपयोग प्रोग्राम को बनाने और साहित्य मे किया जाता है।

Elementary component of Predicate Logic:-

1. Predicate Symbol:
 - a. *Connectives* \neg (not), \vee (or), \wedge (and), \rightarrow (implication)
 - b. *Quantifiers* \exists (there exist/someone), \forall (for all)
2. *Constant symbol*: Fixed value (marcus, 12)
3. *Variable symbol*: Assumed value (x, y, z)
4. *Function symbol*: operational name (f, g, h, mother, father, age etc)

Eg. John's mother married to John's father.

FOPL: $MARRIED(father(John), mother(John))$

Function: $father, mother$

Constant: $John$

Example for representing knowledge in predicate logic:

Sentances	FOPL
1. Marcus was a man.	$Man(marcus)$
2. Marcus was a Pompeian.	$Pompeian(marcus)$
3. All Pompeians were Romans.	$\forall x: Pompeian(x) \rightarrow Roman(x)$
4. Caesar was a ruler.	$Ruler(Caesar)$
5. All romans were either loyal to Caesar or hated him.	$\forall x: Roman(x) \rightarrow Loyalto(x, Caesar) \vee hated(x, Caesar)$
6. Everyone is loyal to someone.	$\forall x: \exists y: Loyalto(x, y)$
7. People only try to assassinate rulers they are not loyal to	$\forall x: \forall y: people(x), \wedge ruler(y) \wedge tryassassinate(x, y) \rightarrow \neg loyalto(x, y)$
8. Marcus tried to assassinate caesar.	$Tryassassinate(marcus, caesar)$

Resolution & question answering:

To prove a valid statement in predicate logic, resolution attempts to show that the negation of the statement produces a contradiction with the known statement. This approach contrasts with the technique that we have been using to generate proofs by chaining backward from the theorem to be proved to the axioms. Using resolution we can find answer of questions.

रेसोल्यूशन एक ऐसा तरीका है जिसमें किसी प्रेडिकेट लॉजिक के स्टेटमेंट को वैध सिद्ध करने के लिए सबसे पहले स्टेटमेंट को नकारात्मक बनाकर हल करते हैं जो एक विरोधाभासक ज्ञात स्टेटमेंट को उत्पन्न करता है। यह धारणा उस तकनीक के विरोधरूप में कार्य करती है जहां हम किसी axiom के प्रूफ से पुनः axiom की ओर अर्थात् पीछे की तरफ चैन बनाकर सिद्ध करते हैं। रेसोल्यूशन के द्वारा पूछे गए प्रश्न का उत्तर भी ज्ञात कर सकते हैं।

Algorithm: Resolution

Let F = a set of given statement, P = a statement to be proved

1. Convert F into clauses form.
2. Negate P and convert the result in clause form.
3. Add result into set of F .
4. Repeat following three point until a contradiction
 - a. Select two clause. (parent clause)
 - b. Resolve them together
 - c. If resolvent is the empty clause, then contradiction has been found.

Algorithm: convert into clause form

1. Eliminate \rightarrow convert $a \rightarrow b$ into $\neg a \vee b$
2. Reduce \neg convert $\neg(a \wedge b)$ into $\neg a \vee \neg b$

For example: resolution proof of question “Is Marcus hate casesar?” its answer will yes if we get nil at the end of proof chain otherwise answer will be no.

Solution:

Convert all set of statement in clause form (F)=

1. $Man(marcus)$
2. $Pompeian(marcus)$
3. $\forall x: Pompeian(x) \rightarrow Roman(x)$ Convert to $\neg Pompeian(x) \vee Roman(x)$
4. $Ruler(Casesar)$
5. $\forall x: Roman(x) \rightarrow Loyalto(x, Caesar) \vee hated(x, Caesar)$ Convert to $\neg Roman(x) \vee Loyalto(x, c) \vee hated(x, c)$
6. $\forall x: \exists y: Loyalto(x, y)$ Convert to $Loyalto(x, y)$
7. $\forall x: \forall y: people(x) \wedge ruler(y) \wedge tryassassinate(x, y) \rightarrow \neg Loyalto(x, y)$
 $\neg [people(x) \wedge ruler(y) \wedge tryassassinate(x, y)] \vee [\neg Loyalto(x, y)]$ Convert to $\neg people(x) \vee \neg ruler(y) \vee \neg tryassassinate(x, y) \vee \neg Loyalto(x, y)$
8. $Tryassassinate(marcus, caesar)$

Negate proof: “Marcus hate casesar” in clause form and into F .

9. $\neg hate(m, c)$

Artificial Intelligence

Now resolve as

Resolve result

using the old sentence with number

$\neg \text{hate}(m,c)$

$\neg \text{Roman}(x) \vee \text{loyalto}(x,c) \vee \text{hated}(x,c)$ -5

$\neg \text{Roman}(m) \vee \text{loyalto}(m,c)$

$\neg \text{Pompeian}(x) \vee \text{Roman}(x)$ -3

$\text{loyalto}(m,c) \neg \text{Pompeian}(m)$

$\text{Pompeian}(\text{marcus})$ -2

$\text{loyalto}(m,c)$

$\neg \text{people}(x) \vee \neg \text{ruler}(y) \vee \neg \text{tryassassinate}(x,y) \vee \neg \text{loyalto}(x,y)$ -7

$\neg \text{man}(m) \vee \neg \text{ruler}(c) \vee \neg \text{tryassassinate}(m,c)$

$\text{Man}(m)$ -1

$\neg \text{ruler}(c) \vee \neg \text{tryassassinate}(m,c)$

$\text{ruler}(c)$ -4

$\neg \text{tryassassinate}(m,c)$

$\text{tryassassinate}(m,c)$ -8

nil

When resolution process stops then we conclude that statement $\neg \text{hate}(m,c)$ "Marcus hate casesar" is true.

www.LRsir.net

Artificial Intelligence

Semantic Nets:

In a semantic net, information is represented as a set of nodes connected to each other by a set of labeled arcs, which represent relationship among the nodes. The main idea behind semantic net is that the meaning of a concept comes from the ways in which it is connected to other concepts.

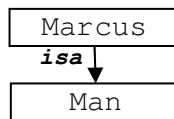
Semantic नेट में जानकारी को कई सारी नोड के रूप में व्यक्त किया जाता है। प्रत्येक नोड एक दूसरे से लेबल आर्क द्वारा कनेक्ट रहती है। तथा ये लेबल आर्क, नोड्स के बीच संबंधों को व्यक्त करती है। सीमेंटीक नेट के पीछे सबसे मुख्य विचार यह है कि, एक से अधिक कान्सेप्ट जो कनेक्टेड है, उन्ही के द्वारा दिये गए कान्सेप्ट का अर्थ प्राप्त किया जाता है।

Example:

Marcus is a Man.

$PL \rightarrow Man(Marcus)$

$SN \rightarrow isa (Marcus, Man)$



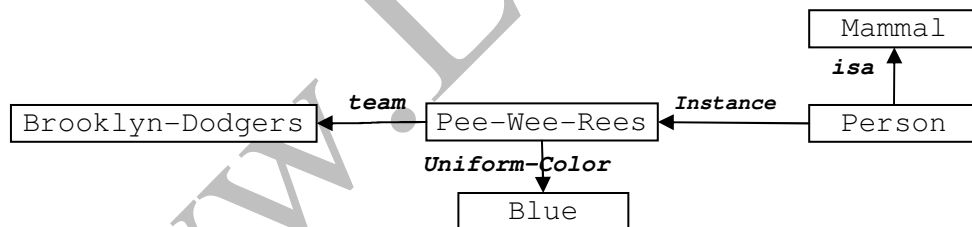
Example:

isa(Person, Mammal)

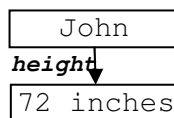
instance(Pee-Wee-Reese, Person)

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

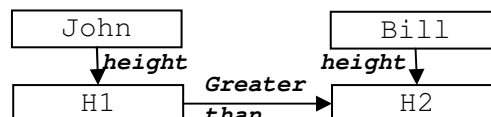
uniform-color(Pee-Wee-Reese, Blue)



Example: *John's height is 72 inches.*



Example: *John is taller than Bill.*



Frames:

A frame is a collection of attributes (usually called slots) and associated values that describe some entity in the world. Sometimes a frame describes an entity in some absolute sense; some times it represents the entity from a particular point of view. A single frame taken alone is rarely useful. Instead, we build frame systems out of collections of frames that are connected to each other by virtue of the fact that the value of an attribute of one frame may be another frame.

वर्ल्ड की कुछ एंटीटीस का वर्णन करने वाले attributes(स्लॉट) और संबन्धित वैल्यूस के समूह को फ्रेम कहा जाता है। किसी एंटीटी को एक फ्रेम के द्वारा सही से समझाया जा सकता है। कई बार फ्रेम के द्वारा एंटीटी के किसी विशेष प्रकार को भी व्यक्त कर सकते हैं। केवल एक ही फ्रेम को बनाकर उपयोग करने के स्थान पर, कई सारी फ्रेम के समूह द्वारा एक फ्रेम सिस्टम का निर्माण करना अधिक उपयोगी होता है। इस प्रकार एक फ्रेम के गुणो(virtue) को किसी अन्य में भी उपयोग कर सकते हैं।

Example:

Person

isa: Mammal
cardinality: 6,000,000,000
**handed:* Right

Adult-Male

isa: Person
cardinality: 2,000,000,000
**height:* 5-10f

ML-Baseball-player

isa: Adult-male
cardinality: 624
**height:* 6-10f
**bats:* equal to handed
**batting-average:* .252
**team:*
**uniform-color:*

Fielder

isa: ML-Baseball-player
cardinality: 376
**batting-average:* .262

Pee-Wee-Rees

isa: Fielder
cardinality: 624
height: 5-10f
bats: Right
batting-average: .309
team: Brooklyn-Dodgers
uniform-color: Blue

ML-Baseball-Team

isa: Team
cardinality: 26
**team-size:* 24
**manage:*

Brooklyn-Dodgers

instance: ML-Baseball-Team
team-size: 24
**manage:* leo-Durocher
Players: {pee-wee-rees,.....}

In this example of frames, *person*, *Adult-male*, *ML-baseball-player*, *fielder* and *ML-baseball-team* are all classes. The frames *pee-wee-reese* and *Brooklyn-dodgers* are instances.

फ्रेम के इस उदाहरण में *person*, *Adult-male*, *ML-baseball-player*, *fielder* और *ML-baseball-team* ये सभी क्लाससेस हैं जबकि *pee-wee-reese* और *Brooklyn-dodgers* की फ्रेमसेस instances हैं।

WWW.LRSir.net

Conceptual Dependency: - (CD)

Conceptual dependency is a theory of how to represent the kind of knowledge about events that is usually contained in natural language sentences.

Conceptual dependency एक ऐसा सिद्धांत है जो यह बताता है कि, किस प्रकार किसी नैचुरल लैंग्वेज में वर्णित एवेंट्स को नॉलेज के एक प्रकार में प्रदर्शित किया जाए।

The goal is to represent the knowledge in a way that-

- Facilitates drawing inferences from the sentences.
- Is independent of the language in which the sentences were originally stated.

CD का मुख्य उद्देश्य नॉलेज को निम्न प्रकार से व्यक्त करने से है।

- वाक्यों से अनुमान लगाने की सुविधा प्रदान करना है।
- यह भाषा पर निर्भर नहीं होती जिसमें वाक्यों को मुख्य रूप से लिखा गया हो।

In CD, representations of actions are built from a set of primitive acts.

CD में विभिन्न एक्शन को व्यक्त करने के लिए निम्न प्रकार के primitive acts के समूह का उपयोग किया जाता है।

ATRNS	used for <i>give</i>
PTRANS	used for <i>go</i>
MOVE	used for <i>movement of a body</i>
INGEST	used for <i>eat</i>
MTRANS	used for <i>tell</i>
MBUILD	used for <i>decide</i>
ATTEND	used for <i>listen/focus on thing</i>

Example: English sentences can be written in CD form as

S = Customer
W = Waiter
C = Cook
M = Cashier
O = Owner

English sentences	CD form
1. Customers are going into restaurant	S PTRANS S into restaurant.
2. Customer focuses eyes to table	S ATTEND eyes to tables.
3. Customer decides where to sit.	S MBUILD where to sit.
4. Customer goes to table	S PTRANS S to table.
5. Customers are moving to sitting position	S MOVE S to sitting position
6. Customer go menu to other customer	S PTRANS menu to S
7. Customer tell signal to waiter	S MTRANS signal to W

Script:

A script is a structure that describes a stereotyped sequence of events in a particular context. A script consists of a set of slots. Associated with each slot may be some information about what kinds of values it may contain as well as a default value to be used if no other information is available.

स्क्रिप्ट एक ऐसा स्ट्रक्चर होता है जो किसी भी इवेंट को एक के बाद एक क्रमशः किसी विशेष प्रसंग में वर्णन करता है। एक स्क्रिप्ट कई सारे स्लॉट से मिलकर बनती है। प्रत्येक स्लॉट, जो आपस में जुड़े हुये हों, उनमें इवेंट से संबंधित कुछ ना कुछ जानकारियाँ अवश्य रहती हैं। यदि स्लॉट में कुछ नहीं हो तब डिफ़ॉल्ट वैल्यू का उपयोग भी किया जा सकता है।

Scripts are useful because, in the real world, there are patterns to the occurrence of events. These patterns arise because of causal relationships between events. Agents will perform one action so that they will then be able to perform another. The events described in a script form a giant *causal chain*.

स्क्रिप्ट बहुत उपयोगी होती है क्योंकि रियल वर्ल्ड में इवेंट के प्राप्त होने के कई सारे तरीके हो सकते हैं। ये सभी तरीके इसलिए उत्पन्न होते हैं क्योंकि सभी एवेंट्स में कुछ न कुछ अनोपचारिक संबंध जरूर होता है। किसी इवेंट के लिए व्यक्ति एक क्रिया को समपन करता है जिससे की वह अन्य क्रिया को करने में समर्थ हो सके। सभी एवेंट्स को एक स्क्रिप्ट में वर्णन करते हैं जो बहुत बड़ी चैन का रूप धारण कर लेती हैं।

The beginning of chain is the set of entry conditions which enables the first events of the script to occur.

चैन की शुरुआत विभिन्न प्रतिबंधों के समूह से होती है जो प्राप्त होने वाली पहली एवेंट्स को स्क्रिप्ट में जोड़ने लायक बनती है।

The end of chain is the set of results which may enable later events or event sequences to occur. Within the chain, events are connected both to earlier events that make them possible and to latter events that they enable.

इस प्रकार चैन का अंत विभिन्न परिणामों का एक समूह हो जाता है जो बाद में होने वाली एवेंट्स या पहले ही चुकी एवेंट्स के क्रम को जोड़ने लायक हो सके। चैन में, पुरानी और बाद की सभी एवेंट्स को जोड़ा जा सकता है।

Use of script:

1. It provides the ability to predict events that have not explicitly been observed.

स्क्रिप्ट की सहायता से ऐसी भी एवेंट्स का अनुमान लगाया जा सकता है जिन्हें बाह्य रूप से निरीक्षण नहीं किया गया हो।

2. It provides a way of building a single coherent interpretation from a collection of observations.

यह विभिन्न निरीक्षणों के समूह से एक ऐसा सुसंगत प्रस्तुतीकरण को प्रदान करने का तरीका प्रदान करती है।

3. It focuses attention on unusual events.

यह अनुपयोगी एवेंट्स पर भी ध्यान आकर्षित करती है।

Following example of script shows part of a typical script, the restaurant script. It illustrates the important components of a script.

Artificial Intelligence

<p>Script: RESTAURANT Track: Cofee Shop Props: Table Menu F=Food Check Money</p>	<p>Scene 1: Entering</p> <p>S PTRANS S into restaurant. S ATTEND eyes to tables. S MBUILD where to sit. S PTRANS S to table. S MOVE S to sitting position</p>
<p>Roles: S = Customer W= Waiter C = Cook M = Cashier O = Owner</p>	<p>Scene 2: Ordering</p> <p>(Menu on table) (W brings menu) (S asks for menu)</p> <p>S PTRANS menu to S S MTRANS signal to W W PTRANS W to table S MTRANS 'need menu' to W</p> <p style="text-align: center;">W PTRANS W to table W ATRANS menu to S</p> <p>S MTRANS W to table *S MBUILD choice of F S MTRANS signal to W W PTRANS W to table S MTRANS 'I want F' to W</p> <p style="text-align: center;">W PTRANS W to C W MTRANS(ATRANS F) to C</p> <p>C MTRANS 'no F' to W C DO(prepare F script) to Scene3 W PTRANS W to S W MTRANS 'no F' to S (go back to *) or (go to Scene 4 at no pay path)</p>
<p>Entry Conditions:</p> <p>S is hungry. S has money.</p>	<p>Scene 3: Eating</p> <p>C ATRANS F to W W ATRANS F to S S INGEST F (Option: Return to Scene 2 to order more; Otherwise, goto Scene 4)</p>
<p>Results:</p> <p>S has less money. O has more money. S is not hungry. S is pleased (optional)</p>	<p>Scene 4: Exiting</p> <p style="text-align: center;">S MTRAN to W (W ATRANS check to S)</p> <p>W MOVE (write check) W PTRANS W to S W ATRANS check to S S ATRANS tip to W S PTRANS S to M S ATRANS money to M</p> <p>(no pay path:) S PTRANS S to out of restaurant</p>

Non monotonic reasoning:

Traditional systems based on predicate logic are monotonic. In monotonic systems there is no need to check for inconsistencies between new statements and the old knowledge. Non monotonic reasoning is based on default reasoning or “most probabilistic choice”.

For example when we visit a friend’s home, we buy biscuits for the children because we believe that most children like biscuits. In this case we do not have information to the contrary.

Default reasoning (or most probabilistic choice) is defined as follows:

Definition 1 : If X is not known, then conclude Y.

Definition 2 : If X can not be proved, then conclude Y.

Definition 3: If X can not be proved in some allocated amount of time then conclude Y.

It is to be noted that the above reasoning process lies outside the realm of logic. It conclude on Y if X can not be proved, but never bothers to find whether X can be proved or not.

This leads to a non monotonic system in which statements can be deleted as well as added to the knowledge base. When a statement is deleted, other related statements may also have to be deleted.

Non monotonic reasoning systems may be necessary due to any of the following reasons.

- The presence of incomplete information requires default reasoning.
- A changing world must be described by a changing database.
- Generating a complete solution to a problem may require temporary assumptions about partial solutions.

Thus non – monotonic systems require more storage space as well as more processing time than monotonic systems.

Statistical and Probabilistic reasoning:

So far we have considered something being ‘[true](#)’ or ‘ not true ‘ or ‘not known’ . There is also the situation ‘probably true’. In situations where “the relevant world is random” or “appears to be random because of [poor](#) representation or “not random but our program can not access large data base”, probabilistic reasoning is to be applied.

(Example of such situations are – motion of electron, a drug being successful on a patient etc.). One has to apply probabilistic reasoning in deciding about the next card to play in a game of cards or in diagnosing the illness from the [symptoms](#). These are random world. Uncertainties can arise from an [inability](#) to predict outcomes due to unreliable, vague, incomplete or inconsistent knowledge.

UNIT-3

LISP AND AI PROGRAMMING LANGUAGES:

- Introduction to LISP :
- Numeric Functions
- Basic List Manipulation Functions in LISP
- Predicates and Conditionals
- Input, Output and Defining Functions
- Iteration and Recursion
- Property List and arrays

PROGLOG and Other AI Programming Languages

Brief overview of LISP:

History: LISP is one of the oldest computer programming languages. It was invented by “John McCarthy” during the late 1950s after FORTRAN.

लिस्प सबसे पुरानी प्रोग्रामिंग भाषा में से एक है। इसे “जॉन McCarthy” के द्वारा फ़ोरट्रान के बाद 1950 में निर्मित किया गया था।

Different Flavors of LISP: Several dialects of LISP are FRANZLISP, INTERLISP, MACLISP, QLISP, SCHEME and COMMON LISP.

लिस्प के कई सारे रूप हैं जैसे FRANZLISP, INTERLISP, MACLISP, QLISP, SCHEME और COMMON LISP.

Important Features of LISP: It is suited for AI programming because of its ability to process symbolic information effectively. LISP has simple syntax with little or no data typing and dynamic memory management.

लिस्प AI प्रोग्रामिंग के लिए सबसे अधिक उपयुक्त मनी जाती है क्योंकि इसमें सिंबोलिक से संबंधित जानकारियों को अच्छे से प्रोसेस करने की छमता होती है। लिस्प में सबसे कम अर्थात् नहीं के बराबर डाटा को टाइप करने वाले और डाइनेमिक मेमोरी को मैनेजमेंट करने वाले सरल सिंटेक्स पाये जाते हैं।

Running the LISP program: LISP program run on an interpreter or as compiled code. The interpreter examines source programs in a repeated loop (read-evaluate-print). This loop reads LISP program code, evaluate it, and print the value returned by the program. LISP interpreter read code using -> prompt.

लिस्प का प्रोग्राम इंटरप्रेटर या compiled कोड के द्वारा रन होता है। इंटरप्रेटर सोर्स प्रोग्राम को एक रिपिटेड लूप(read-evaluate-print)में परीक्षण करता है। यह लूप लिस्प प्रोग्राम के कोड सबसे पहले रीड करता है फिर evaluate कर प्रोग्राम के द्वारा लोटाइ गई वैल्यू को प्रिंट करता है। लिस्प इंटरप्रेटर -> प्रॉम्प्ट के द्वारा कोड को रीड करता है।

Ex:

-> (+ 5 6 7)

19

-> Input here next lisp instruction.

Basic Building of LISP: (atom, list and string)

Atom, list and string are the valid object in LISP, also called “ Symbolic Expressions or S-expressions”.

लिस्प के मान्य ऑब्जेक्ट Atom, list और string है। इन्हे “ Symbolic Expressions या S-expressions” के नाम से भी जाना जाता है।

Atom: A number, continuous character (include digits, alphabets, special characters).

कोई भी संख्या, निरंतर क्रम वाले कैरक्टर (include digits, alphabets, special characters) को एटम कहा जाता है।

Ex: Numbers: 2013, lokesh, this-is-a-atom, ab123

List: A sequence of atoms and/or other lists enclosed within parentheses () and separated by space. Elements of list are called top element of list.

Atoms एवं/या अन्य लिस्ट का क्रमबद्ध समूह जिन्हे छोटे कोष्ठक () में लिखा जाए और प्रत्येक एक स्पेस के द्वारा प्रथक हो तब इसे लिस्ट कहते हैं। लिस्ट के एलेमेंट को top element of list कहा जाता है।

Ex: (this is a list), (a (a b) c (def)), (mon tue wed thus fri sat sun), () empty list

String: A group of characters enclosed in double quotation marks.

विभिन्न कैरक्टर का एक समूह जो डबल कोट “...” में बंद हो उन्हें स्ट्रिंग कहते हैं।

Ex: “this is a string”

Special Value:

There are three types under special values-

स्पेशल वैल्यूस के अंतर्गत तीन प्रकार की वैल्यूस होती हैं।

- 1) Constant: any number like 12, 199
- 2) t: for true and
- 3) nil: for false or () empty list.

Function call:

Every operation is called function and applying content (atom, list, string) called argument. Everything will be written in list form and function-name written as prefix.

लिस्प में प्रत्येक ऑपरेशन को फंक्शन कहा जाता है तथा अप्लाई किए जाने वाले कंटेंट (atom, list, string) को आर्गुमेंट कहते हैं। सभी को यानहा एक लिस्ट के रूप में लिखा जाता है और फंक्शन-नेम को प्रेफिक्स में लिखते हैं।

Syntax for function call :

(function-name arg1 arg2 -----)

Ex: (+ 2 4) + is a function

When a function is called, the arguments are first evaluated from left to right and function is executed using the evaluated argument values.

जब किसी फंक्शन को कॉल किया जाता है तब सबसे पहले लेफ्ट से राइट की ओर आर्गुमेंट जात किए जाते हैं और फंक्शन को जात किए गए आर्गुमेंट की वैल्यू के द्वारा रन किया जाता है।

Syntax and Numeric function : (+,-,*,/)

Arithmetic function operate only passing numeric arguments (integer or real value).

Arithmetic function केवल पास की जाने वाली numeric arguments (integer or real value) को ही हल करते हैं।

Syntax: -> (op data-1 data-2 ----- data-n)

Op means (+ or – or * or /)

1) **+** (plus): Add zero or more given arguments.

-> (+ 2 4 6) gives 12

2) ***** (product): Multiply zero or more given arguments.

-> (* 2 4) gives 8

3) **-** (minus): Subtract two given arguments.

-> (- 4 1) gives 3

4) **/** (Divide): Divide two given arguments.

-> (/ 9 3) gives 3

Declaring Variable (setq):

Used to hold data (atom/ list) for further use and return last bound data. Unbound variable gives error.

वेरियबल का उपयोग data (atom/ list) को स्टोर करने के लिए किया जाता है जिससे उनका उपयोग बाद में किया जा सके।

Syntax: setq(variable-name bound-data)

Ex:

-> (setq x 10)

10

->x

10

Basic List Manipulation function: (car, cdr, cons and list):

Used to extract data from list .

इनका उपयोग लिस्ट में से डाटा की छटनी करने के लिए किया जाता है।

1) **Car:** Returns top element of given list.

यह फंक्शन दी गई लिस्ट के टॉप element को लौटाता है।

Example-

-> (car '(a b c))

Output: a

2) **Cdr:** Returns a list except first top element of list.

यह फंक्शन दी गई लिस्ट के टॉप element को छोड़कर शेष सभी एलेमेंट को लौटाता है।

Example-

-> (cdr '(a b c))

Output: (b c)

3) **Cons:** This function need only two arguments, an element (atom or list) and a list, as the result returns a list with the element inserted at the beginning of list.

यह फंक्शन दो आर्गुमेंट्स (atom or list) और एक लिस्ट पर कार्य करता है। परिणामतः पहले आर्गुमेंट को सेकंड लिस्ट में सबसे आगे जोड़ देता है।

Example-

-> (cons 'a '(b c))

Output: (a b c)

Element a add at top of list (b c)

4) **List:** This function is used to create a list for given a number of elements as arguments. It returns a list.

यह फंक्शन आर्गुमेंट्स के रूप में दिये गए सभी एलिमेंट्स को एक लिस्ट के रूप में निर्मित कर देता है।

Example-

-> (list 'a '(b c) 'd)

Output: (a (b c) d)

Create a list contain a,(b c) and d.

Predicate functions:

Predicates are function that test their arguments for some specific condition and return true (t) or false (nil).

प्रेडिकेट्स वे फंक्शन होते हैं जो इनपुट किए गए आर्गुमेंट्स को एक विशेष कंडिशन के लिए परीक्षण करते हैं और true(t) या false(nil) वैल्यू को प्रदान करते हैं।

The most common predicates are:

सबसे अधिक उपयोग किए जाने वाले predicates निम्न हैं।

1) **atom:** It needs one argument, if argument is an atom then returns t(true) otherwise nil(false).

यदि दिया गया आर्गुमेंट एक एटम है तब t(ट्रू) अन्यथा nil वैल्यू प्राप्त होती है।

Ex:

-> (atom 'a) output: t

-> (atom '(a)) output: nil

2) **listp:** It needs one argument, if argument is a list then returns t(true) otherwise nil(false).

यदि दिया गया आर्गुमेंट एक लिस्ट है तब t(ट्रू) अन्यथा nil वैल्यू प्राप्त होती है।

Ex:

-> (listp '(a)) output: t

-> (list 'a) output: nil

- 3) **evenp**: It needs one argument, if argument is a even number then returns t(true) otherwise nil(false).

यदि दिया गया आर्गुमेंट एक ईवन नंबर हे तब t(ट्रू) अन्यथा nil वैल्यू प्राप्त होती है।

Ex:

-> (evenp 8) output: t

-> (evenp 9) output: nil

- 4) **oddp**: It needs one argument, if argument is a odd number then returns t(true) otherwise nil(false).

यदि दिया गया आर्गुमेंट एक ऑड नंबर हे तब t(ट्रू) अन्यथा nil वैल्यू प्राप्त होती है।

Ex:

-> (oddp 9) output: t

-> (oddp 8) output: nil

The conditional (cond function):

If we want to perform any action based on given condition then it is possible using cond function in LISP. It is like the if...then...else construct. यदि

किसी एक्शन को दी गई कंडिशन के आधार पर सम्पन्न करना हो तब लिस्प मे यह कार्य cond फंक्शन की सहायता से पूरा करते है। यह if -then -else सरचना के समान होता है।

Syntax :

(cond (<test₁> <action₁>) (<test₂> <action₂>).....(<test_k> <action_k>))

Features of cond:

- Cond function can test one or more conditions.
- If <test₁> has t(true) then <action₁> portion is performed otherwise control passes to the next part.

Cond फंक्शन एक या अधिक कंडिशनस को टेस्ट कर सकता है। यदि <test₁> t है तो <action₁> सम्पन्न होगा अन्यथा कंट्रोल अगले पार्ट पर पास हो जाता है।

Ex: Test ginen number is even or odd.

-> (cond ((evenp 8) 'Even) (t 'odd)))

Even

-> (cond ((evenp 7) 'Even) (t 'odd)))

Odd

Input function: (read)

Read takes no arguments. When read appears in a procedure, processing halts until a single s-expression is entered from the keyboard.

किसी प्रोसीजर में Read फंक्शन बिना किसी आर्गुमेंट के कीबोर्ड से एंटर की गई वैल्यू को प्राप्त कर सकता है।

For example:

-> (+ 5 (read))

6

11

Assign to variable:

->(setq x (read))

8 (enter)

8 (output, 8 has assigned to x)

-> x (enter)

8 (value of x)

Output function

Print function can print argument from beginning of new line.

Print फंक्शन इनपुट आर्गुमेंट को न्यू line से प्रिंट करता है।

For example:

->(print '(hello))

Hello (by print and return to interpreter)

Hello (by interpreter)

Defining Functions:

It is possible to user define functions in LISP. i.e. user can define own function according to his requirement. In LISP this task is performed using predefine defun function.

लिस्प में यूजर अपनी आवश्यकतानुसार defun फंक्शन के द्वारा खुद का भी फंक्शन परिभाषित कर सकता है।

Syntax:

->(defun udfn(p₁ p₂ ...p_n) body)

Here-

defun= function maker

udfn= user define function name

p= list of parameter which receive values

body= single line of logic in list form.

Ex1: function for average of three numbers.

-> (defun averagethree(n1 n2 n3) (/ (+ n1 n2 n3) 3))

Output: AVERAGETHREE

Now we call this function as usual-

-> (averagethree 10 20 30)

Output: 20

Ex2: Convert centigrade to Fahrenheit.($f=c*9/5+32$)

->(defun ctof(c) (+ (/ (* c 9) 5) 32))

Output:ctof

-> (ctof 0)

Output: 32

Iteration and recursion:

Iteration is one form of loop whereas recursion is a special kind of user function that call ownself. .

किसी भी लूप से संबन्धित कार्य को **iteration** कहते हैं जबकि विशेष प्रकार के यूसर फंक्शन जो खुद को ही कॉल करे उसको **recursion** कहते हैं।

Iteration:

When a set of statements are executed more than one times using a loop until a given condition satisfied called iteration. using do function we can create an iteration.

जब किसी लूप के द्वारा स्टेटमेंट के एक समूह को एक या अधिक बार रन किया जा सके जब तक की दी गई कंडिशन संतुष्ट नहीं हो जाती तब यह कार्य **iteration** कहलाता है।

Do फंक्शन का उपयोग **iteration** बनाने के लिए किया जाता है।

Syntax:

```
(do (<var1 val1> <var-update1>
    (<var2 val2> <var-update2>)
    .
    .
    (<test> <return-value>)
    (<s-expression>))
```

Here:

- <val> are initial values that bounds to <var> variables parallel first time then <test> is evaluated.
- If<test> gives nil value then all <var> are parallel updated with <var-update> value. This process continued until test get non-nil(true).
- If <test> get non-nil any time then iteration process stops and it return <return-value>.
- <s-expression> is optional. If present, executed with each iteration

यंहा:

- सबसे पहली बार variable <var> मे वैल्यू <val> स्टोर होती है इसके बाद <test> हल होता है।
- यदि <test> की वैल्यू nil हो तब सभी <var> मे एक साथ <var-update> वैल्यू सेट होती हे। यह प्रोसेस निरंतर चलती है जब तक की कभी <test> non-nil(t) प्रदान नहीं कर देता ।
- <s-expression> का यूस वैकल्पिक है। यह लूप के साथ प्रत्येक iteration पर रन होता है।

Ex: factorial function

```
->(defun factorial(n)
  ( do
    (i n (- i 1))
    (f n (* f (- i 1))
      ((equal 1 i) f)
    )
  )
)
```

Factorial

```
-> (factorial 4)
```

24

Description: Initially value 4 assign to i and f, since i is not 1 so test give nil and second times i update by 3(- i 1) and f updated by 12(* f (- i 1)). After some iteration i updated by 1 then test give non-nil and iteration will terminate with f value.

वर्णन: सबसे पहले वैल्यू 4 factorial फंक्शन के आर्गुमेंट n में पास होगी उसके बाद do लूप के वैरियबल i और f में एक साथ स्टोर होगी। test पार्ट में चूंकि i की वैल्यू 1 नहीं है इसलिए variable i की वैल्यू (- i 1) अर्थात् 3 से और f की वैल्यू (* f (- i 1)) अर्थात् 12 से अपडेट होगी। कुछ iteration होने के बाद i को 1 से अपडेट होगा और test पार्ट non-nil देगा और iteration factorial की वैल्यू को f के द्वारा लौटा देगा।

Recursion:

It is a special kind of function in which all statements of function are executed more than one times until any condition satisfied, but it will called recursion if every times all statements relocated and previous will remains in memory. Simply when a function calls itself until a given condition called recursion.

यह फंक्शन का एक विशेष प्रकार होता है जिसमें फंक्शन के सभी स्टेटमेंट एक से अधिक बार दी गई कंडिशन के संतुष्ट होने तक रन होते रहते हैं। किन्तु ऐसे दोहराव को तभी recursion कहा जाएगा यदि सभी स्टेटमेंट मेमोरी में बार बार रिलोकेट हों और पिछले वही स्टेटमेंट भी मेमोरी में बने रहे।

सरल शब्दों में जब कोई फंक्शन खुद को ही एक निश्चित सीमा तक कॉल करे तब इसे recursion कहते हैं।

Ex: factorial function using recursion

```
->(defun factorial(n)
  (cond ( (equal n 1) 1)
        (t (* n (factorial(- n 1))))))
```

Factorial

```
-> (factorial 4)
```

24

Initially value 4 assign to n, since n is not 1 so this function again call with one less value. When it get 1 then recursively recalculate.

Artificial Intelligence

वर्णन: सबसे पहले वैल्यू 4 आर्गुमेंट n में स्टोर होती है फिर जब तक की n की value 1 नहीं हो जाती यह फंक्शन खुद को ही n में 1 को घटाकर कॉल करने लगता है। जब n की वैल्यू 1 हो जाती है तब यह recursively फक्टोरियल की वैल्यू को ज्ञात करता है।

Array:

Collection of items that stores continuously called array. To create and access array we use following three function.

एक समान items का समूह जो निरंतर क्रम में स्टोर हो उन्हें array कहते हैं। लिस्प में निम्न प्रकार के फंक्शन की सहायता से array को बनाकर उसका उपयोग करते हैं।

- 1) make-array: It is used to create a new blank array of given size in memory.

Syntax:

```
(setf arr-name (make-array '(size)))
```

Ex:

```
->(setf myarray(make-array '(5)))
```

Output: #A(nil,nil,nil,nil,nil)

- 2) aref: It is used to access any specific array item.

Syntax:

```
(aref arr-name index)
```

Index= 0 to size-1

Ex: Assign values to array

```
-> (setf (aref myarray 0) 10)
```

Output: 10

```
-> (setf (aref myarray 4) 50)
```

Output: 50

Ex: Read array values

```
-> (aref myarray 0)
```

Output: 10

```
-> (aref myarray 4)
```

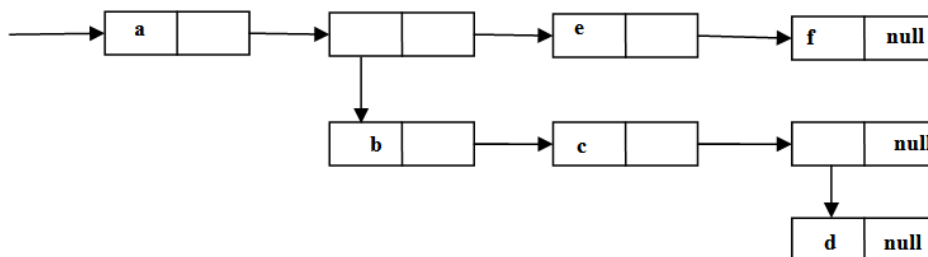
Output: 50

Internal storage of list:

Lists are made through the use of linked cell structures in memory.

लिस्प में बनाई जाने वाली लिस्ट मेमोरी में एक लिंक लिस्ट के रूप में स्टोर होती है।

For example: (a (b c (d))) e f) can be represented as-



PROLOG and other AI programming languages

History: PROLOG(PROgramming in LOGic) was invented by “Alain Colmerauer” and his associates at the university of Marseilles during the early 1970s.

PROLOG(PROgramming in LOGic) को “अलाइन कोल्मेरौएर” और उसके साथियों के द्वारा 1970 के पहले मरसेल्लेस यूनिवर्सिटी में बनाया गया था।

Resolution process of PROLOG:

It uses the syntax of predicate logic to perform symbolic, logical computations. Programming in PROLOG is accomplished by creating a database of facts and rules about objects, their properties, and their relationships to other objects. Queries can then be posed about the objects and valid conclusions will be determined and returned by the program. Responses to user queries are determined through a form of inferencing control known as resolution.

प्रोलॉग में symbolic और लॉजिकल computations को सम्पन्न करने के लिए predicator लॉजिक का उपयोग किया जाता है। प्रोलॉग में प्रोग्रामिंग को करने के लिए ओब्जेक्ट्स से संबंधित फेक्ट्स, रूल्स, प्रॉपर्टीस और अन्य ओब्जेक्ट्स से सम्बन्धों का एक डेटाबेस तैयार किया जाता है। ओब्जेक्ट्स से संबंधित queries को जमाया जाता है, फिर valid निष्कर्ष को ज्ञात कर बनाए गए प्रोग्राम के द्वारा लोटा देते हैं। यूजर के प्रश्नों का जवाब एक interfacing कंट्रोल form की सहायता से प्रस्तुत करते हैं जिसे रेसोल्यूशन कहते हैं।

Facts representation in PROLOG:

Facts are declared with predicates and constants written in lowercase letters. The arguments of predicates are enclosed in parentheses and separated with commas(,).

प्रोलॉग में फेक्ट्स को प्रेडिकेट्स के साथ डिक्लैर करते हैं और कॉन्स्टेंट्स को छोटे लेटर्स में लिखते हैं। प्रेडिकेट्स के आर्गुमेंट्स को छोटे कोष्ठक () में लिखकर कोमा (,) द्वारा separate करते हैं।

Ex: some facts about family relationship can be written in PROLOG as-

उदा. प्रोलॉग में फैमिली से संबंधित कुछ फेक्ट्स को निम्न प्रकार से लिखा जा सकता है।

- 1) sister(sue, bill) - sue is sister of bill.
- 2) parent(ann, sam) - ann is parent of sam.
- 3) parent(joe, ann) - joe is parent of ann.
- 4) male(joe) - joe is a male.
- 5) female(ann) - ann is a female.

Here sister, parent, male and female are predicate, whereas sue, bill, ann and sam are arguments.

यहां sister, parent, male और female प्रेडिकेट हैं, जबकि sue, bill, ann और sam आर्गुमेंट्स हैं।

Rules in PROLOG:

Rules are composed of a condition or “if” part and a conclusion or “then” part separated by a symbol (:-). Rules are used to represent general relations which hold when all of the conditions in the if part are satisfied. Rules may contain variables(uppercase).

प्रोलॉग में रूल्स को तैयार करने के लिए कंडिशन अर्थात “if” पार्ट और निष्कर्ष अर्थात “then” पार्ट के द्वारा बनाते हैं। कंडिशन और निष्कर्ष :- के द्वारा प्रथक रहते हैं। रूल्स का उपयोग सामान्य संबंधों को व्यक्त करने के लिए करते

Artificial Intelligence

है यदि सभी कंडिशनस if पार्ट में संतुष्ट होती है। रूल्स में variables भी हो सकते हैं जिनमें uppercase में लिखा जाता है।

Ex: PROLOG rules for grand father, we write

उदा. ग्रांड फादर के लिए प्रोलॉग रूल्स को निम्न प्रकार से लिखा जा सकता है।

Grandfather(X,Z):- parent(X,Y), parent(Y,Z), male(X)

This rule has following meaning-

For all X, Y and Z,

X is the grand father of Z

If X is the parent of Y, and Y is the parent of Z and X is the male.

Query in PROLOG:

When a database of facts and rules such as that above have given, then we make queries by typing after (?) symbol such as-

जब उपरोक्तानुसार फेक्ट्स और रूल्स के द्वारा डेटाबेस दिया गया हो तब (?) सिम्बल के ठीक बाद टाइप कर queries बनाते हैं।

?- parent(X, sam) -who is parent of sam

X=ann -Answer

?- male(joe)

Yes

?- grandfather(X,Y)

X=hoe, Y=sam

?- female(joe)

No

PROLOG searches database for submitted query. If a proper match found between predicates of query and database then returns response with proper result or failure occurs.

प्रोलॉग सबमिट की गई क्वेरी के लिए डेटाबेस को सर्च करता है। यदि क्वेरी के प्रेडिकेट और डेटाबेस के बीच प्रोपर मैच प्राप्त होता है तब सही या गलत परिणाम के साथ response मिलता है।

Lists in PROLOG:

Each item must be separated by commas(,) and enclosed in square brackets[].

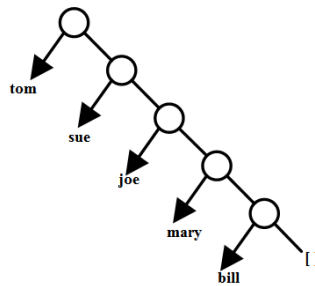
प्रोलॉग में प्रत्येक आइटम कोमा(,) द्वारा प्रथक रहता है तथा सभी स्क्वियर ब्रेकेट [] में बंद होते हैं।

Ex: [tom, sue,joe,mary,bill] is the list of student in PROLOG.

A list may be empty or non empty. A non empty list has head and tail. Tom is the head and remaining sublist [sue,joe,mary,bill] is tail.

लिस्ट empty या नॉन empty हो सकती है। नॉन empty लिस्ट में एक हैड और शेष टेल होता है। Tom हैड है और शेष सभी टेल हैं।

Binary tree representation of PROLOG list:



List may be written as $[a,b,c,d]=[a,b|[c,d]]=[a,b,c,d|[]]$.

List manipulate predicates:

A number of list manipulation predicates are also available in PROLOG as- प्रोलॉग में भी कई सारे लिस्ट मैनिपुलेशन प्रेडिकेट्स उपलब्ध होते हैं जैसे-

append, member, conc, add, delete and so on.

For example:

?- member(c,[a,b,c,d])

Yes

?- member(b,[a,[b,c],d])

no

PROLOG has numeric functions, relations and list handling capabilities.

प्रोलॉग में न्यूमेरिक फंक्शन, रिलेशन्स और लिस्ट को हंडल करने की क्षमता होती है।

Other programming languages used in AI:

C, object oriented extensions to LISP such as Flavors, and languages like Smalltalk. The language C has been used by some AI programmers because of its popularity and its portability. Although Object oriented languages have been gaining much popularity.

प्रोलॉग एवं LISP के अलावा और भी कई अन्य AI languages होती हैं जैसे- सी भाषा, LISP की ऑब्जेक्ट ओरिएंटेड विस्तारित भाषाएँ जैसे Flavour और Smalltalk। C भाषा की प्रसिद्धि और पोर्टबिलिटी की कारण कुछ AI प्रोग्रामर इसका उपयोग कर चुके हैं। यद्यपि ऑब्जेक्ट ओरिएंटेड भाषाओं का उपयोग AI प्रोग्रामिंग में सबसे अधिक किया जाता है।

UNIT-4

Introduction to Expert System

Structure of an Expert system

Interaction with expert

Design of an expert system.

Introduction to Expert System:

A Knowledge-based expert system use human knowledge to solve problems that normally would require human intelligence. Expert systems are designed to carry the intelligence and information found in the intellect of experts and provide this knowledge to other members of the organization for problem solving purposes.

A Knowledge-Based Expert System किसी भी problem को solve करने के लिए मनुष्य के knowledge का उपयोग करता है जिसमे सामान्य रूप से human intelligence की आवश्यकता होती है। एक्सपर्ट सिस्टम को डिज़ाइन करने के लिए किसी एक्सपर्ट पर्सन से जुड़कर उसके इंटेलिजेंस नेचर एवं जानकारीयों को प्राप्त किया जाता है तथा उसके knowledge को organization के किसी अन्य मेम्बर को problem सॉल्व करने के लिए प्रदान कर सकते हैं।

Basic Meaning of Expert System:

Expert system is one of the areas of artificial intelligence. An expert system also known as knowledge based system. It is a computer program that contains the knowledge and analytical skills of one or more human experts in a specific problem domain.

Expert सिस्टम भी Artificial Intelligence का एक भाग है। Expert सिस्टम को नॉलेज बेस्ड सिस्टम के नाम से भी जाना जाता है। यह एक ऐसा कम्प्युटर प्रोग्राम होता है जिसमे किसी problem domain के लिए, एक या अधिक ह्यूमन एक्सपर्ट के नॉलेज और उनके analytical skill को शामिल करते हैं।

The goal of the design of the expert system is to capture the knowledge of a human expert in such a way that it can be available to a less experienced user.

एक्सपर्ट सिस्टम को बनाने का मुख्य उद्देश्य यह होता है की किसी इसके द्वारा किसी ह्यूमन एक्सपर्ट के नॉलेज को भी कम या अनुभवहीन यूजर को भी उपलब्ध करवाया जा सकता है।

Expert system provides high quality experience, domain specific knowledge; apply heuristics, forward or backward reasoning, uncertainty and explanation capability.

एक एक्सपर्ट सिस्टम उच्च गुणवत्ता के साथ उससे संबन्धित ही नॉलेज को प्रदान करता है; इसके लिए इसमे कई सारी मेथड को अप्लाई करते हैं जैसे- heuristics, forward या backward रीज़निंग, uncertainty तथा वर्णन करने की क्षमता आदि ।

Characteristics of an Expert System:

- 1) The most important ingredient in any expert system is the knowledge.
एक्सपर्ट सिस्टम का सबसे मुख्य घटक नॉलेज है।
- 2) In expert systems, knowledge is separated from its processing i.e. the knowledge base and the inference engine are split up.

Artificial Intelligence

एक्सपर्ट सिस्टम में, नॉलेज को उसकी प्रोसेसिंग एक्टिविटी से अलग रखा जाता है अर्थात् इसके मुख्य दो भाग- Knowledge Base एवं Inference engine दोनों अलग-अलग होते हैं।

- 3) Expert system contains a knowledge base having accumulated experience and a set of rules for applying the knowledge base to each particular situation that is described to the program.

एक्सपर्ट सिस्टम में एक Knowledge Base होता है जिसमें सारे अनुभवों को एकत्रित करते हैं एवं उस knowledge base पर वे सभी rule अप्लाई करते हैं जो प्रोग्राम में वर्णित हैं।

- 4) Expert system provides the high-quality performance which solves difficult programs in a domain as good as or better than human experts.

किसी डोमेन से संबंधित परेशानियों को हल करने में ह्यूमन एक्सपर्ट की तुलना बनाया गया एक्सपर्ट सिस्टम उच्च-गुणवत्ता के साथ प्रस्तुति देता है।

- 5) Expert systems employ symbolic reasoning when solving a problem. Symbols are used to represent different types of knowledge such as facts, concepts and rules.

जब किसी problem को हल करना हो तो एक्सपर्ट सिस्टम, symbolic रीजनिंग पर कार्य करता है। Symbols अलग-अलग प्रकार के नॉलेज का उपयोग करते हैं जैसे facts(तथ्य), concepts एवं rules।

Interaction with Expert:

Expert is a person that has fully knowledge about subject of matter. In the absence of that person, we want to work according to that expert. Therefore we need to develop an intelligence program that reasoning in the same way of that expert and contain his whole life of knowledge, called Knowledge Base(KB-Set of If Condition Then Knowledge).

एक्सपर्ट एक ऐसा व्यक्ति होता है जिसे किसी विषय से संबंधित सम्पूर्ण ज्ञान होता है। ऐसे व्यक्ति की अनुपस्थिति में, हम उसी के अनुसार कार्य करना चाहते हैं। इसलिए हमें एक intelligence प्रोग्राम को developed करना होगा जिसकी reasoning बिल्कुल एक्सपर्ट जैसी ही हो एवं उसमें एक्सपर्ट की लाइफ का सम्पूर्ण नॉलेज उपलब्ध हो जिसे नॉलेज बेस(KB) के नाम से जाना जाता है।(KB- If Condition Then Knowledge का समूह)

The knowledge base elicited from the expert by a trained knowledge engineer using various methods can include methodical interviews and the repertory grid technique. Often the expert knowledge area is "fuzzy" in nature and contains a great deal of procedural knowledge, so the knowledge engineer must be an expert in the process of knowledge elicitation.

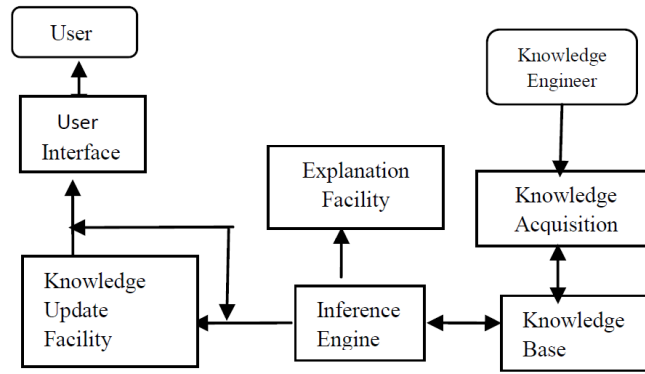
एक trained knowledge इंजीनियर के द्वारा किसी एक्सपर्ट से Knowledge Base को प्राप्त किया जाता है। इसके लिए वह कई सारी मेटड की सहायता लेता है जैसे- methodical interview और repertory grid तकनीक मुख्य हैं। कुछ एक्सपर्ट नॉलेज का एरिया "fuzzy" प्रकार का होता है एवं उसमें प्रोसीजर नॉलेज को डील करना होता है, इसलिए नॉलेज इंजीनियर को भी नॉलेज हासिल करने की प्रोसेस में expert होना चाहिए।

ARCHITECTURE OF AN EXPERT SYSTEM:

Expert system can be built using a piece of development software known as a 'shell'. The core components of expert systems are the knowledge base and the reasoning engine.

एक्सपर्ट सिस्टम को डेवलपमेंट सॉफ्टवेयर के विभिन्न पहलुओं की सहायता से निर्मित किया जाता है जिन्हें शैल कहा जाता है। एक्सपर्ट सिस्टम का मुख्य घटक Knowledge Base और reasoning engine होता है।

Artificial Intelligence



Knowledge Base:

It is a warehouse of the domain specific knowledge captured from the human expert via the knowledge acquisition module. To represent the knowledge production rules, frames, logic, semantic net etc. is used.

एक्सपर्ट सिस्टम का यह एक ऐसा भाग होता है जिसे knowledge Acquisition module के द्वारा ह्यूमन एक्सपर्ट के domain specific knowledge को प्राप्त कर एक warehouse तैयार करते हैं। इसका उपयोग knowledge production rule, frames, logic, semantic आदि में करते हैं।

Inference Engine:

Inference Engine is a brain of expert system. It uses the control structure (rule interpreter) and provides methodology for reasoning. The major task of inference engine is to trace its way through a forest of rules to arrive at a conclusion. Here two approaches are used i.e. forward chaining and backward chaining.

Inference Engine को एक्सपर्ट सिस्टम का brain कहा जाता है। यह कंट्रोल स्ट्रक्चर का उपयोग करती है और reasoning के लिए एक methodology प्रदान करती है। Inference Engine का मुख्य कार्य forest of rule की सहायता से एक ऐसा तरीका बनाना जिसके द्वारा किसी निष्कर्ष पर पहुंचा जा सके। इस कार्य के लिए दो approach अपनाई जा सकती है- forward chaining एवं backward chaining ।

Knowledge Acquisition:

Knowledge acquisition is the accumulation, transfer and transformation of problem-solving expertise from experts and/or documented knowledge sources to a computer program for constructing or expanding the knowledge base. For knowledge acquisition, techniques used are protocol analysis, interviews, and observation.

Knowledge Acquisition वह होता है जिसमें किसी एक्सपर्ट person/ किसी लेख के knowledge source को कम्प्यूटर प्रोग्राम में इकट्ठा कर ट्रान्सफर किया जाता है जिससे की एक Knowledge Base को बनाया जा सके या बढ़ाया जा सके। Knowledge Acquisition के लिए विभिन्न तकनीकों को उपयोग में लाया जाता है जैसे-प्रोटोकॉल एनालिसिस, interview एवं observation.

Explanation Facility:

It is a subsystem that explains the system's actions. Here user would like to ask the basic questions why and how and serves as a tutor in sharing the system's knowledge with the user.

यह एक सब सिस्टम होता है जो सिस्टम की क्रियाओं का वर्णन करता है।

Artificial Intelligence

User interface:

It provides facilities such as menus, graphical interface etc. to make the dialog user friendly. Responsibility of user interface is to convert the rules from its internal representation (which user may not understand) to the user understandable form.

To build the expert system is known as Knowledge Engineering. The expert and knowledge engineer should anticipate user's need while designing an expert system.

Application of Expert System:

Expert system can be applicable in many areas. Some more are-

- 1) Different type of medical diagnosis.
- 2) Diagnosis of complex electronic and electromechanical systems.
- 3) Diagnosis of diesel electric location systems.
- 4) Diagnosis of software development projects.
- 5) Planning experiments in biology, chemistry and molecular genetics.
- 6) Forecasting crop damage.
- 7) Identification of chemical compound structures and chemical compounds.
- 8) Location of faults in computer and communications system.
- 9) Evaluation of loan applicants for lending institution.
- 10) Analysis of structural systems for design or as a result of earth quake.

Early application area of expert systems -

- 1) DENDRAL(1960): First Expert System which recognizes the structures of chemical compounds.
- 2) MYCIN: which diagnoses bacterial blood infections.
- 3) PUFF: which diagnose pulmonary disorders.
- 4) SCHOLAR: which gives Geography Tutorials.
- 5) SOPHIE: which teaches how to detect breakdown in electrical circuits.
- 6) SHDRU: which manipulates polygons in a restricted environment.
- 7) Waterman's Poker Player: Game playing systems.
- 8) AM: Automatic theorem Provers.
- 9) NOAH and MOLGEN: Planning systems.
- 10) Holland: Prediction systems such as Political Forecasting Systems.

Importance of Expert System:

The value of expert systems was well established by early 1980s. A number of successful applications had been completed by then and they proved to be cost effective. An example which illustrates this point well is the diagnostic system developed by the Campbell Soup Company.

एक्सपर्ट सिस्टम की वास्तविक वैल्यू को 1980 के पहले एस्टाब्लिश किया गया था। उसके बाद बहुत सारी एप्लिकेशन को सफलता पूर्वक बनाया गया और उन सभी ने कोस्ट इफेक्टिव को सिद्ध भी किया जा चुका है। एक्सपर्ट सिस्टम का एक उदाहरण प्रस्तुत है जिसके diagnostic सिस्टम को कैम्पबेल सोप कंपनी के द्वारा निर्मित किया गया था।

Campbell Soup uses large cookers to cook soups and other canned products at eight plants located throughout the company. For the maintenance of cooker fault, only a single human expert was to diagnosis. He had to flying all site when necessary. Since this individual will retire after some year then company decided computer based expert system to diagnosis cooker fault problem.

Artificial Intelligence

कैम्पबेल सोप कंपनी 8 प्लांट्स में सूप और उसके अन्य प्रकार के **canned** प्रोडक्ट्स बनाने के लिए बहुत बड़े कूकर का उपयोग होता था। कूकर के फ़ाल्ट को दुरुस्त करने के लिए केवल एक ही ह्यूमन एक्सपर्ट था। वह आवश्यकतानुसार सभी साइट्स पर जाता था। चूंकि वह अकेला एक्सपर्ट कुछ साल बाद रिटाइर होने वाला होगा तब कंपनी ने एक कम्प्युटर पर आधारित एक्सपर्ट सिस्टम को बनाने का निर्णय लिया जो कूकर फ़ाल्ट की प्रोब्लेम की पहचान कर सके।

After some month, Texas Instruments, developed an Expert System used to provide training to new maintenance personal.

कुछ महीनो बाद टेक्सास इन्स्ट्रुमेंट्स कंपनी ने भी एक एक्सपर्ट सिस्टम बनाया जिसका उपयोग न्यू मैटेनेंस पर्सनल को ट्रेनिंग देने में किया जाता था।

Thus computer expert system never retire, so its very great importance in many areas.

अतः कम्प्युटर एक्सपर्ट सिस्टम कभी रिटाइर नहीं होता, इसलिए एक्सपर्ट सिस्टम का महत्व कई सारे areas में हो सकता है।

WWW.LRSir.net

UNIT-5

Neural Network:

Basic structure of neuron

Perception

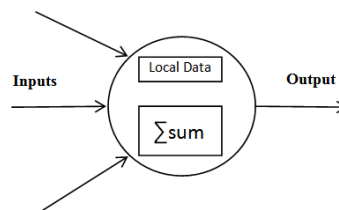
Feed forward and back propagation

Hopfield network

Artificial Neural Network(ANN) or Neural network(NN):

ANN is a network of many simple processor (unit/neuron/node) each possibly having a small amount of local memory. The neurons are connected by unidirectional communication channels (connection), which carry numeric data. The neurons operate only on their local data and inputs that receive via the connection.

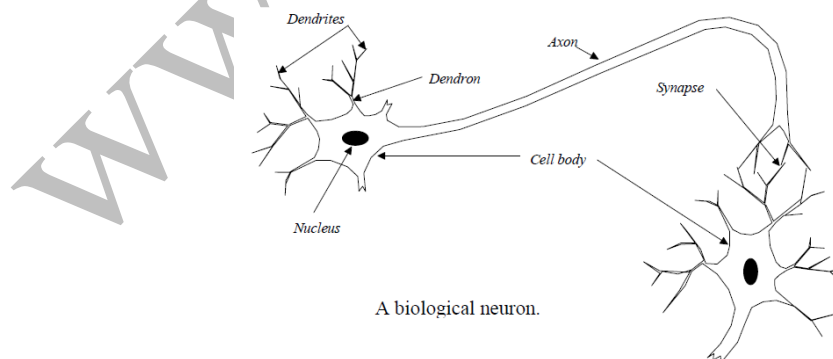
ANN कई सारे simple processor(unit/neuron/node) का एक network है जंहा प्रत्येक में संभवतया एक small लोकल मेमोरी होती है। इस नेटवर्क में सभी न्यूरॉन्स unidirectional चैनल के द्वारा कन्नेक्टेड रहते हैं जिंका उपयोग numeric डाटा को मुव किया जा सकता है। न्यूरॉन्स केवल कनेक्शन पर किए गए इनपुट डाटा तथा लोकल डाटा को ही ऑपरेट करते हैं।



Biological Neurons:

Designs and functionality of NN are based on human's brain structures.

मनुष्य के दिमाग की संरचना के आधार पर Neural नेटवर्क को डिज़ाइन एवं कार्य व्यवहार में लाया जाता है।



The human brain contains about 10 billion nerve cells (neurons). Each neuron is connected to the others through 10000 synapses. A neuron has a branching input (dendrites), a branching output (the axon). Axon connects to dendrites via synapses. The information circulates from the dendrites to the axon via the cell body. In this way brain can learn, reorganize itself from experience.

Artificial Intelligence

किसी भी मनुष्य के brain में लगभग 10 billion nerve सेल्स(न्यूरॉन्स) होते हैं। प्रत्येक न्यूरॉन लगभग 10000 synapses की सहायता से connected होती है। प्रत्येक न्यूरॉन में एक इनपुट(dendrites) ब्रांच, एक आउटपुट(axon) ब्रांच होती है। synapses की सहायता से Axon को dendrites से कनेक्ट किया जाता है। समस्त जानकारी dendrites से axon तक सेल बॉडी के द्वारा घूमती रहती है। इसप्रकार से ह्यूमन brain लगातार अनुभव के आधार पर सीखता रहता है।

According to Dr. Robert Nielsen- “A computing system made up of a number of simple, highly interconnected processing elements which process information by dynamic state responses to external input known as Neural Network (NN).

डॉ रोबर्ट नीलसेन के अनुसार- “ एक ऐसा कम्प्यूटिंग सिस्टम जो बहुत ही साधारण तथा उच्च-श्रेणी के interconnected प्रोसेसिंग एलिमेंट्स से मिलकर बना हो, जिनका मुख्य कार्य बाहरी इनपुट को बदलती हुई अवस्था के साथ जानकारी प्रदान करना हो ”, तब ऐसी संरचना NN कहलाती है।

Most NNs are based on training rule where values (or weight) that inputs at connections are adjusted for present pattern. In other words NN learn just like a child.

बहुत सारे NN ट्रेनिंग रूल पर आधारित होते हैं जिसमें connection पर इनपुट की जाने वाली values(या weight) को present pattern के लिए एडजस्ट किया जाता है। दूसरे शब्दों में NN एक छोटे बच्चे जैसे ही सीखता है।

Neurons are often known as elementary non linear single processor. NN is different than other computing devices, because NN has a high degree of interconnection which allows parallelism.

न्यूरॉन्स को अक्सर elementary non linear single processor के नाम से भी बोला जाता है। NN किसी अन्य computing devices से बिल्कुल भिन्न होते हैं, क्योंकि NN में सबसे उच्च श्रेणी का interconnection जो parallelism को मान्य करता है।

Further NN has non ideal memory contain data, program but rather each neuron in program continuous active.

इसके अलावा NN में एक सक्रिय मेमोरी भी होती है जिसमें डाटा तथा प्रोग्राम को रखा जाता है किन्तु यह अनिवार्य नहीं है की प्रोग्राम के लिए सभी न्यूरॉन्स active रहे।

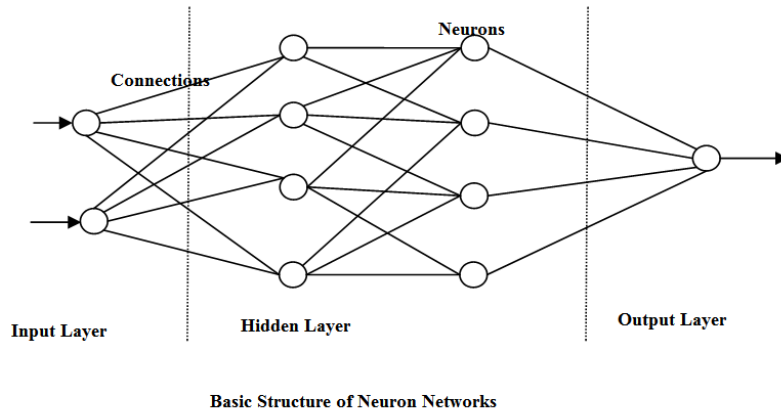
Basic structure of Neural Networks:

NN are typically organized in many layers and each layer is consists using a number of interconnected neurons which contain 'activation function'. Structure of NN is dividing into three layers.

NN जैसे तो कई सारी layers में organized होता है एवं प्रत्येक लेयर प्रत्येक लेयर कई सारी संख्या में न्यूरॉन्स से inter कनेक्टेड रहते हैं जिसमें 'activation function' मौजूद हो। NN की संरचना तीन layers में विभक्त की गई है।

- 1) Input Layer
- 2) Hidden Layer
- 3) Output Layer

Artificial Intelligence



Descriptions: (वर्णन)

Input Layer:

Any patterns are presented to network using Input layers.

Input layers की सहायता से किसी भी pattern को network में प्रस्तुत करते हैं।

Hidden Layer:

Input patterns are communicated to one or more hidden layer where the actual processing is done via a system of weighted connection.

Input patterns एक या अधिक hidden layer के द्वारा communicate करता है जंहा सिस्टम के कनेक्शन पर रखे गए weight के द्वारा वास्तविक प्रोसेसिंग की जाना है।

Output Layer:

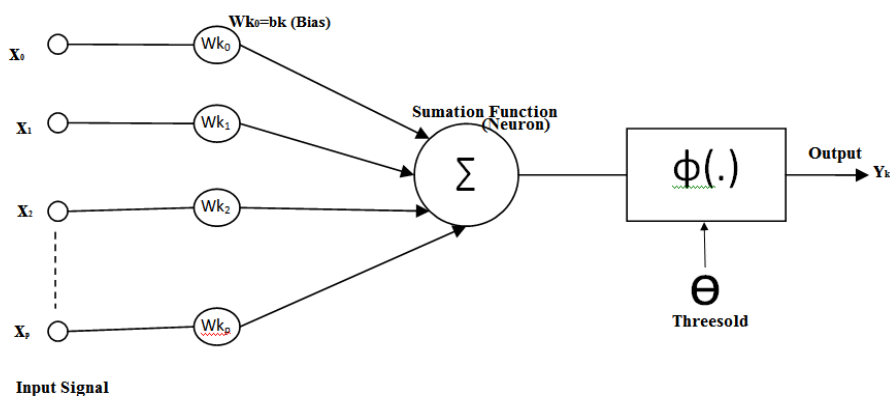
When hidden layer is link to an output layer then it present answers of given pattern.

जब hidden लेयर किसी आउटपुट लेयर से लिंक होती है तब यह दिये गए पैटर्न का answer प्रस्तुत करती है।

Most ANN contains some form of a learning rule which modify a weight of the connection according to input pattern.

कई सारे ANN में लर्निंग रूल के विभिन्न रूप होते हैं जिसमें इनपुट पैटर्न के आधार पर connection के weight में परिवर्तन किया जाता है।

The mathematical structural model of neurons:



Input Value= $x_0, x_1, x_2, \dots, x_p$

Weight Value= $Wk_0, Wk_1, x_2, \dots, Wk_p$

Internal activity of neuron is-

$$V_k = \sum_{j=1}^p Wk_j * x_j$$

Output of Neuron is –

$$Y_k = V_k + \theta$$

(some activation function value)

Thus an artificial neuron consists of inputs which are multiply by weights and then computed by a mathematical function. In this way it determines the activation of neuron.

अतः एक artificial neuron कई सारे inputs से मिलकर बनाता है जिसे एक mathematical function की सहायता से connection line पर adjust किए गए weight से multiply किया जाता है। इस प्रकार से यह neuron के activation value को ज्ञात करता है।

Application of NN: Pattern reorganization, predication, system identification and control.

Advantages of ANN:

- 1) A Neural Network can perform such task which is not possible in linear programming.
एक neural network ऐसे कार्यों को भी सम्पन्न कर सकता है जो किसी linear programming के द्वारा संभव है।
- 2) When a neuron of NN fails then it can continue without any problem.
जब NN का कोई neuron कार्य करना बंद कर देता है तब भी यह neural network बिना किसी परेशानी के निरंतर कार्य कर सकता है।
- 3) All neurons are always learn, so need not to be reprogrammed.
NN में सभी neurons हमेशा सीखते रहते हैं, इसलिए इन्हें बार बार program नहीं करना होता।
- 4) It can be implemented in any application.
इन्हें किसी भी application में उपयोग कर सकते हैं।
- 5) It can be implemented without any problem.
NN को बिना किसी परेशानी के implement कर सकते हैं।

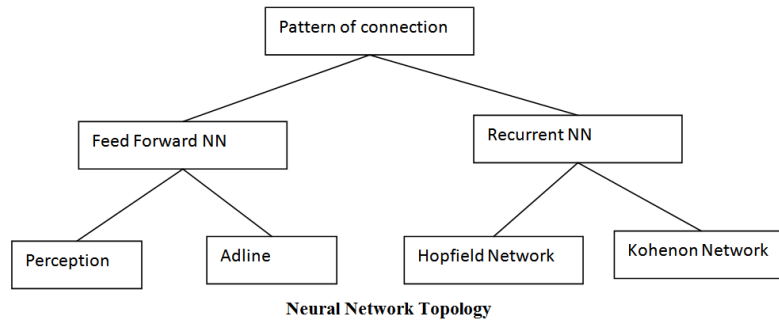
Disadvantages:

- 1) NN needs training to operate.
NN को operate करने के लिए विशेष training की आवश्यकता होती है।
- 2) Structure of NN is different than microprocessor therefore NN needs always to reconstruct.
NN का structure microprocessor से पूरी तरह से भिन्न होता है इसलिए NN को हमेशा reconstruct करने की आवश्यकता बनी रहती है।
- 3) Requires high processing time for large networks.
बहुत बड़े network के लिए सबसे अधिक processing time की आवश्यकता होती है।

Neural Network Topology:

The pattern of connection between neurons and propagation of data called NN topology. Different NN topologies can be classified into following ways.

Neurons एवं data के propagation के बीच connection के pattern को NN topology कहा जाता है। विभिन्न NN topologies को निम्न प्रकार से वर्गीकृत किया जा सकता है।



Feed Forward NN:

The Feed Forward NN was the first and simplest type of ANN where data flow from input neurons to output neurons strictly feed forward.

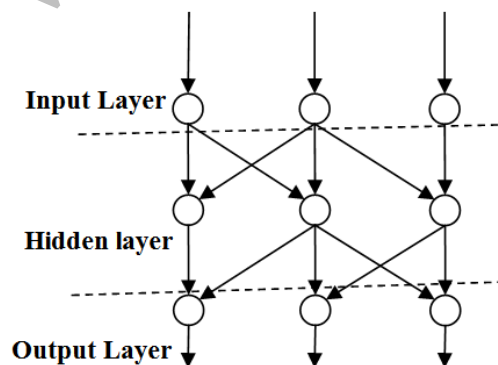
Feed Forward NN सबसे पहला और सरलतम ANN के प्रकारों में से एक रहा है। इसमें इनपुट न्यूरॉन्स से आउटपुट न्यूरॉन्स की ओर flow होने वाले data अनिवार्यतः आगे की ओर feed(भरना) करते हुये बढ़ते जाते हैं।

In this network the information move in only one direction forward from input neurons through the hidden layer neurons(if any) and to output neurons.

इस नेटवर्क में जानकारीया केवल एक ही दिशा में आगे की ओर इनपुट न्यूरॉन्स से होते हुये hidden layers के न्यूरॉन्स में होती हुई आउटपुट न्यूरॉन्स की ओर बढ़ती है।

Feed Forward NN can be represented as:

Feed Forward NN को निम्न प्रकार से प्रदर्शित किया जा सकता है-



There is no any loop or cycle in the network through which information can move to visited neurons again. In feed forward NN the data processing can extend over multiple layers of neuron but no feedback connections are present. That is, there is no way to moves output layer to input layers.

Artificial Intelligence

Example of Feed Forward NN: Perception, Adline.

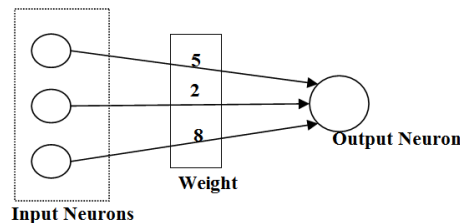
इस नेटवर्क में ऐसी कोई साइकल या लूप निर्मित नहीं होता है जिससे की information visit किए जा चुके न्यूरॉन्स पर पुनः मुव हो सके। Feed Forward NN में data processing को न्यूरॉन्स की कई सारी layers में विस्तारित तो किया जा सकता है किन्तु कोई feedback connection उपलब्ध नहीं होता है। अर्थात इसमें ऐसा कोई तरीका नहीं है जिसके द्वारा आउटपुट layer से input layer की ओर मुव किया जा सके।

Feed Forward NN के उदाहरण: Perception, Adline।

Perception or Perceptron:

This model was proposed by Rosenblatt on 1959. This was first NN which has the ability to learn. In this model inputs neurons typically have two states-On / Off. Whereas output neuron uses a simple threshold activation functions.

NN के इस model को Rosenblatt द्वारा 1959 में प्रस्तावित किया गया था। यह सबसे पहला NN था जिसमें कुछ सीखने की ability थी। इस model में input न्यूरॉन्स को मुख्यतः दो स्टेट- On/Off में रखा गया। जबकि आउटपुट न्यूरॉन केवल एक ही साधारण threshold activation function का उपयोग करता है।



Perceptron can only solve linear problem.

Perceptron केवल linear problem को ही सॉल्व कर सकता है।

Perceptron learning process:

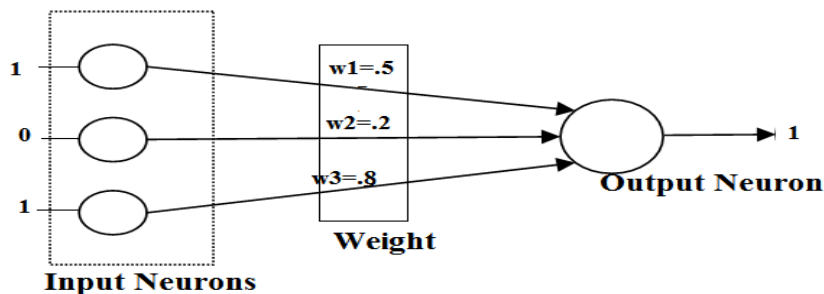
If output is not correct, the weights are adjusted according to following formula.

यदि आउटपुट सही नहीं है जैसा हम चाहते हैं, तब निम्न सूत्र के द्वारा weights को adjust किया जाता है।

$$W_{\text{new}} = W_{\text{old}} + \alpha (\text{desired output} - \text{current output}) * \text{Input}.$$

α is learning rate.

For example-



$$\begin{aligned} \text{Current output} &= (1 * .5) + (0 * .2) + (1 * .8) \\ &= 1.3 \end{aligned}$$

Artificial Intelligence

But we have to assume output value is 0. So to get 0 at output, we will adjust weight.

किन्तु हमें इन्हीं इनपुट्स पर 0 आउटपुट चाहिए इसलिए हमें weight को adjust करना होगा।

Let $\alpha=1$.

Now apply all values to formula for each existing weight to get new weight value.

फॉर्मूले में फिर से न्यू न्यू weight को अप्लाई करते हैं।

$$W_{\text{new}} = W_{\text{old}} + \alpha (\text{desired output} - \text{current output}) * \text{Input}$$

$$W_1 = .5 + 1 * (0 - 1) * 1 = -.5$$

$$W_2 = .2 + 1 * (0 - 1) * 0 = .2$$

$$W_3 = .8 + 1 * (0 - 1) * 1 = -.2$$

Now recalculate output value with new weights.

$$\text{New output} = (1 * -.5) + (0 * .2) + (1 * -.2)$$

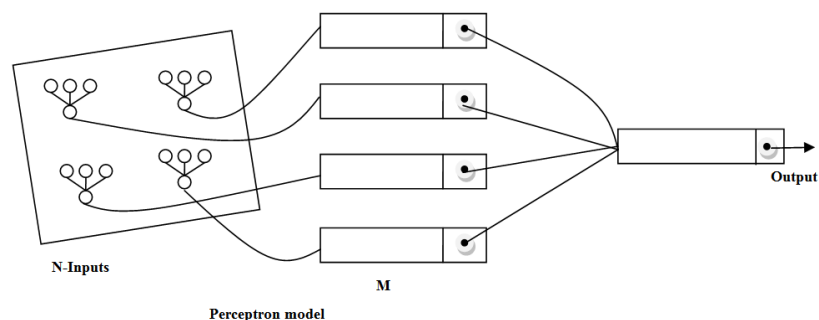
$$= -.5 + 0 + -.2$$

$$= -.7 \text{ (equivalent to 0)}$$

In Perceptron positive outputs are considered 1(one) and negative values are 0(zero).

Perceptron में positive output को 1 एवं negative को 0 माना जाता है।

Perceptron Model



Thus we can say that perceptron model consists of N-element inputs layer which feeds into a layer of M association masks or predicate units (hidden) on a single output unit.

अतः हम कह सकते हैं कि perceptron model, N-element वाले इनपुट्स layer से मिलकर बनता है जो किसी एक आउटपुट यूनिट के लिए उससे उससे जुड़ी हुई M मास्क या predicate units(hidden) को फीड करता है।

The goal of the operation of perception is to learn a given transformation learning samples with input and corresponding outputs.

Perception के operation का मुख्य उद्देश्य किसी दिये गए इनपुट के साथ संगत आउटपुट प्राप्त करने के लिए लर्निंग transformation sample को सिखाना है।

The original meaning of perceptron is that the activity of predicate unit can be any function of inputs layers the learning procedure only adjusts the connection of output unit.

Artificial Intelligence

Perceptron का वास्तविक अर्थ यह है की, जब किसी predicate यूनिट पर input layer के द्वारा कोई कार्य किया जाता है तब learning procedure के अंतर्गत केवल output यूनिट के connection को ही एडजस्ट किया जाता है। तब ऐसी एकटिविटी वाले यूनिट/न्यूरॉन को perceptron कहा जाता है।

Back Propagation:

A feed forward did not present a solution to the problem of “ how to adjust the weight from input unit to hidden unit.” Its solution is back propagation. According to back propagation rule, to determine errors of hidden layer’s neurons, first we determine errors at output layer’s neurons.

“किस प्रकार इनपुट यूनिट से hidden यूनिट के weight एडजस्ट किए जाए” इस प्रोब्लेम के हल को feed forward प्रस्तुत नहीं कर सकी। इसका हल back propagation है। Back propagation के नियम के अनुसार- hidden layers की error को ज्ञात करने के लिए हम सबसे पहले आउटपुट न्यूरॉन पर प्राप्त होने वाली errors को ज्ञात करते हैं।

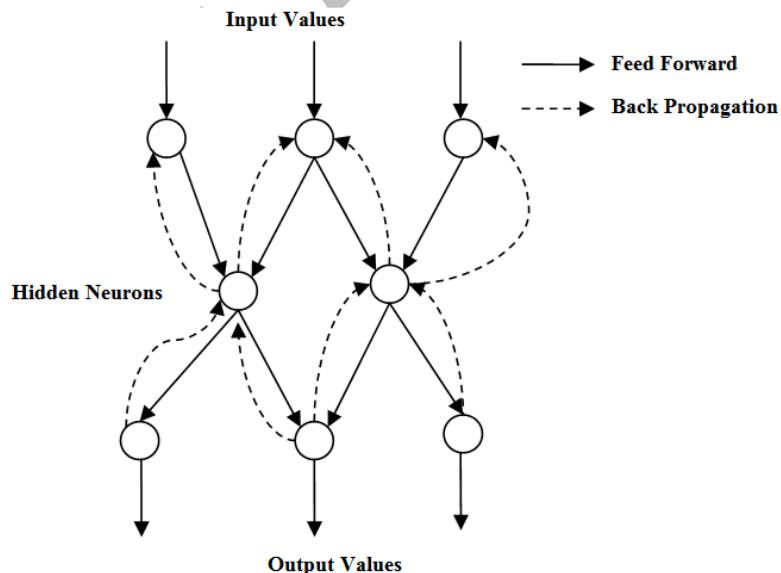
Back propagation can also be considered as-generalization of delta value for non linear function and multilayered network.

Back propagation को non-linear एवं multilayered network की डेल्टा वैल्यू को generate करने वाली मेथड के रूप में माना जा सकता है।

Back-Propagation method:

When a learning pattern is clamped(adjusting weight for desired output), the activation values are propagate towards output neurons and the actual network output is compared with desired.

जब किसी लर्निंग पैटर्न को clamp(चाहे गए आउटपुट के लिए weight को एडजस्ट करना) किया जाता है, तब आउटपुट न्यूरॉन्स की ओर activation वैल्यू propagate(आगे बढ़ना) होने लगती है एवं नेटवर्क के actual आउटपुट को चाहे गए आउटपुट के साथ तुलना करते हैं।



Back propagation performed in two phases.

Back propagation निम्न दो phase में होता है।

Artificial Intelligence

Phase 1: In this phase, input is presented and propagated forward through the network to compute output values for each output neurons. Obtained output is compared with desired output values. We get result an error signal for each output neurons.

इस phase में, सबसे पहले values को इनपुट न्यूरॉन्स में प्रस्तुत कर नेटवर्क के द्वारा हमेशा आगे की दिशा में बढ़ाते रहते हैं जिससे प्रत्येक आउटपुट न्यूरॉन्स के लिए आउटपुट वैल्यू को compute कर सके। प्राप्त आउटपुट को चाहे गए आउटपुट के साथ तुलना करते हैं। हमें परिणाम के रूप में प्रत्येक आउटपुट न्यूरॉन्स पर एक error signal प्राप्त होता है।

Phase 2: In this phase a backward pass through a network during which the error signal is pass to each network and appropriate changed.

इस phase में नेटवर्क के द्वारा एक backward pass किया जाता है जिसमें error signal को प्रत्येक नेटवर्क में pass करते हैं और यथोचित बदलाव करते जाते हैं।

Phase 1 and phase 2 will continued until all error signal removes.

Phase 1 एवं Phase 2 तब तक निरंतर बना रहता है जब तक की सभी error सिग्नल खत्म नहीं हो जाते।

Algorithm: Back Propagation

This algorithm used for adjusting weight when output neuron gets error signal.

यह algorithm उपयोग में लाई जाती है जब आउटपुट पर प्राप्त error सिग्नल को खत्म करने के लिए weight को adjust करना होता है।

Step 1: Present input for the first pattern to the input layer.

इनपुट लेयर पर पहले pattern के लिए इनपुट प्रस्तुत करते हैं।

Step 2: Sum the weighted inputs to the next layer and calculate their activation value.

Next लेयर पर सभी weighted इनपुट का sum करते हैं और उसकी activation value को calculate करते हैं।

Step 3: Present activation to the next layer. Repeat step-2 until the activation of output layer are known.

प्राप्त activation वैल्यू को next लेयर पर प्रस्तुत करते हैं। Step-2 को तब तक repeat करते हैं जब तक की आउटपुट लेयर की एक्टिवेशन वैल्यू प्राप्त नहीं हो जाती।

Step 4: For a given pattern evaluate output activation to get target value, and determine output layer's details.

किसी दिये गए pattern के लिए target वैल्यू को प्राप्त करने हेतु आउटपुट activation को ज्ञात करते हैं, एवं आउटपुट लेयर की डिटेल्स का पता करते हैं।

Step 5: Using output layer's detail, errors are propagate to backward so that delta value can be evaluated for previous layer.

Output लेयर की detail की सहायता से, प्राप्त errors को backward (पीछे की ओर) propagate (बढ़ाना) करते जाते हैं जिससे की पिछली लेयर की डेल्टा वैल्यू को प्राप्त किया जा सके।

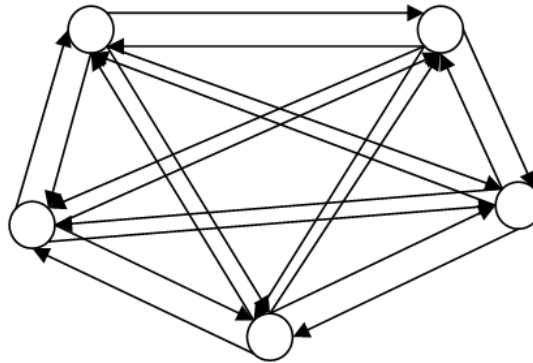
Recurrent Network (Hopfield Network):

A NN which has one or more feedback loops called recurrent network. (feed forward does not contain loop).

एक ऐसा NN जिसमें एक या अधिक feedback loops होते हैं उन्हें recurrent network कहते हैं। (feed forward में ऐसा कोई loop नहीं होता है।)

Hopfield network is one type of recurrent network. It is also called continuous deterministic network model. In Hopfield NN every neuron is connected to all other neurons and connections are symmetric then such connection of neurons called Hopfield model.

Hopfield Network, recurrent नेटवर्क का एक प्रकार होता है। इसे Continuous deterministic network model भी कहा जाता है। Hopfield NN में प्रत्येक neuron अन्य सभी न्यूरॉन्स से कनेक्टेड रहता है और सभी कनेक्शन symmetric होते हैं तब ऐसे न्यूरॉन्स के कनेक्शन को Hopfield model के नाम से जाना जाता है।



This structure shows Hopfield network. The basic model of Hopfield is consist using a number of processing element(neuron) that compute weighted sum of input and threshold to get output of binary value.

यह structure Hopfield network को दर्शाता है। Hopfield model की आधारभूत संरचना कई सारे प्रोसेसिंग एलेमेंट(न्यूरॉन्स) की सहायता से मिलकर बनती है जो weighted इनपुट्स के योग एवं threshold के द्वारा बाइनरी वैल्यू के रूप में आउटपुट को प्राप्त करते हैं।

In Hopfield network all neurons are both inputs and outputs. It means a pattern is clamped on neurons then it iterate on network in looping form. After some time it gets stable state. This state is output of network. In Hopfield obtained output is new activation value of neuron.

Hopfield network में सभी न्यूरॉन्स इनपुट्स एवं outputs दोनों के रूप में व्यवहार कर सकते हैं। इसका अर्थ यह है कि किसी चाहे गए पैटर्न को पहले सभी न्यूरॉन्स पर रखते हैं और उसे लूप के रूप में पूरे नेटवर्क में दोहराते हैं। कुछ समय बाद यह स्थिर अवस्था में आ जाता है। नेटवर्क कि यही अवस्था output कहलाती है। Hopfield में प्राप्त आउटपुट न्यूरॉन्स कि न्यू एक्टिवेशन वैल्यू होती है।

Thus Hopfield network consist of a set of N interconnected neurons which update their activation value asynchronously and each neuron does not depends on other neurons to perform this task, because all neurons of Hopfield are input and outputs. Activation value of each neuron are binary(1/0).

अतः Hopfield network एक दूसरे से जुड़े हुये N-न्यूरॉन्स के समूह से मिलकर बना होता है जो उसकी वैल्यू को asynchronously (अनियमित रूप से) update करता है एवं प्रत्येक न्यूरॉन्स उसके परिणाम के

Artificial Intelligence

लिए अन्य न्यूरॉन पर निर्भर भी नहीं रहता, क्योंकि Hopfield के सभी न्यूरॉन्स इनपुट एवं आउटपुट दोनों ही रूप में कार्य कर सकते हैं। प्रत्येक न्यूरॉन्स कि वैल्यू हमेशा बाइनरि(1/0) के रूप में consider होती है।

Mathematically in Hopfield model the net input of a neuron can be calculated using following formula.

गणितीय रूप में Hopfield model के प्रत्येक न्यूरॉन पर नेट इनपुट को निम्न सूत्र के द्वारा calculate कर सकते हैं-

$$S_k(t+1) = \sum j_i(t)W_{jk} + \theta_k$$

K=a neuron

t= at time

j= input value

W=Weighted value

θ = threshold value

In Hopfield a neuron call stable at time (t)

Hopfield में किसी time t पर एक न्यूरॉन को stable बोल सकते हैं।

This equation gives following any one value.

यह समीकरण निम्न में से किसी एक वैल्यू को प्रदान करती है-

+1 (Stable state)

-1 (Unstable state)

Application of Hopfield network:

Hopfield model is basically used to implement concept of associated memory in which searching is based on given word.

Hopfield model का मुख्यतः उपयोग associative memory कि अवधारणा को लागू करने में किया जाता है जहां searching किसी दिये गए word पर आधारित होती है।