

ARTIFICIAL INTELLIGENCE

Digital Notes By

BIGHNARAJ NAIK

Assistant Professor

Department of Master in Computer Application

VSSUT, Burla

Syllabus

5th SEMESTER MCA

F.M.- 70

MCA-308

ARTIFICIAL INTELLIGENCE (3-1-0)Cr.-4

Module I (10 hrs.)

Introduction to Artificial Intelligence: The Foundations of Artificial Intelligence, The History of Artificial Intelligence, and the State of the Art. Intelligent Agents: Introduction, How Agents should Act, Structure of Intelligent Agents, Environments. Solving Problems by Searching: problem-solving Agents, Formulating problems, Example problems, and searching for Solutions, Search Strategies, Avoiding Repeated States, and Constraint Satisfaction Search. Informed Search Methods: Best-First Search, Heuristic Functions, Memory Bounded Search, and Iterative Improvement Algorithms.

Module II (10 hrs.)

Agents That Reason Logically; A Knowledge-Based Agent, The Wumpus World Environment, Representation, Reasoning & Logic propositional Logic : A very simple Logic, An agent for the Wumpus World.

First-Order Logic; Syntax and Semantics, Extensions and National, Variations, using First Order Logic, Logical Agents for the Wumpus World, A Simple Reflex Agent, Representing Change in the World, Deducing Hidden Properties of the World, Preferences Among Actions, Toward A Goal-Based Agent.

Building a Knowledge Base; Properties of Good and Bad Knowledge Bases, Knowledge Engineering. The Electronic Circuits Domain, General Outology, The Grocery Shopping World. Inference in First-Order Logic : Inference Rules Involving Quantifiers, An Example Proof. Generalized Modus Ponens, Forward and Backward, Chaining & Completeness, Resolution: A complete Inference Procedure, Completeness of Resolution.

Module III (10 hrs.)

Planning A Simple Planning Agent Form Problem Solving to Planning. Planning in Situation Calculus. Basic Representations for Planning. A Partial-Order planning Example, A partial Order planning algorithm, Planning With partially Instantiated Operators, Knowledge Engineering for Planning.

Making Simple Decision: Combining Beliefs and desires under uncertainty. The Basis of Utility Theory, Utility Functions. Multi attribute utility Functions, Decision Networks. The Value of Information. Decision – Theoretic Expert Systems.

Learning in Neural and Belief Networks' How the Brain Works, Neural Networks, perceptions, Multi-layered Feed Forward Networks Applications Back propagation algorithm Applications of Neural Networks.

Module IV (10 hrs.)

Knowledge in Learning: Knowledge in Learning, Explanation-based Learning, Learning Using Relevance Information, Inductive Logic Programming. Agents that Communicate: Communication as action, Types of Communicating Agents, A Formal Grammar for A subset of English Syntactic Analysis (Parsing), Definite Clause Grammar (DCG), Augmenting A Grammar. Semantic Interpretation. Ambiguity and Disambiguation. A Communicating Agent. Practical Natural Language processing Practical applications. Efficient Parsing Scaling up the lexicon. Scaling up the Grammar Ambiguity. Discourse Understanding.

Reference Books:

1. Elaine Rich, Kevin Knight, & Shivashankar B Nair, Artificial Intelligence, McGraw Hill, 3rd ed.,2009
2. Introduction to Artificial Intelligence & Expert Systems, Dan W Patterson, PHI.,2010

MODULE WISE DESCRIPTIONS OF ALL THE CONCEPTS

Module 1:

What is Artificial Intelligence?

Artificial Intelligence (AI) is a branch of *Science* which deals with helping machines finding solutions to complex problems in a more human-like fashion. This generally involves borrowing characteristics from human intelligence, and applying them as algorithms in a computer friendly way. A more or less flexible or efficient approach can be taken depending on the requirements established, which influences how artificial the intelligent behaviour appears. AI is generally associated with *Computer Science*, but it has many important links with other fields such as *Maths, Psychology, Cognition, Biology* and *Philosophy*, among many others. Our ability to combine knowledge from all these fields will ultimately benefit our progress in the quest of creating an intelligent artificial being.

AI currently encompasses a huge variety of subfields, from general-purpose areas such as perception and logical reasoning, to specific tasks such as playing chess, proving mathematical theorems, writing poetry, and diagnosing diseases. Often, scientists in other fields move gradually into artificial intelligence, where they find the tools and vocabulary to systematize and automate the intellectual tasks on which they have been working all their lives. Similarly, workers in AI can choose to apply their methods to any area of human intellectual endeavour. In this sense, it is truly a universal field.

HISTORY OF AI

The origin of artificial intelligence lies in the earliest days of machine computations. During the 1940s and 1950s, AI begins to grow with the emergence of the modern computer. Among the first researchers to attempt to build intelligent programs were Newell and Simon. Their first well known program, logic theorist, was a program that proved statements using the accepted rules of logic and a problem solving program of their own design. By the late fifties, programs existed that could do a passable job of translating technical documents and it was seen as only a matter of extra databases and more computing power to apply the techniques to less formal, more ambiguous texts. Most problem solving work revolved around the work of Newell, Shaw and Simon, on the general problem solver (GPS). Unfortunately the

GPS did not fulfill its promise and did not because of some simple lack of computing capacity. In the 1970's the most important concept of AI was developed known as Expert System which exhibits as a set rules the knowledge of an expert. The application area of expert system is very large. The 1980's saw the development of neural networks as a method learning examples.

Prof. Peter Jackson (University of Edinburgh) classified the history of AI into three periods as:

1. Classical
2. Romantic
3. Modern

1. Classical Period:

It was started from 1950. In 1956, the concept of Artificial Intelligence came into existence. During this period, the main research work carried out includes game plying, theorem proving and concept of state space approach for solving a problem.

2. Romantic Period:

It was started from the mid 1960 and continues until the mid 1970. During this period people were interested in making machine understand, that is usually mean the understanding of natural language. During this period the knowledge representation technique "semantic net" was developed.

3. Modern Period:

It was started from 1970 and continues to the present day. This period was developed to solve more complex problems. This period includes the research on both theories and practical aspects of Artificial Intelligence. This period includes the birth of concepts like Expert system, Artificial Neurons, Pattern Recognition etc. The research of the various advanced concepts of Pattern Recognition and Neural Network are still going on.

COMPONENTS OF AI

There are three types of components in AI

1) Hardware Components of AI

- a) Pattern Matching
- b) Logic Representation
- c) Symbolic Processing
- d) Numeric Processing
- e) Problem Solving
- f) Heuristic Search
- g) Natural Language processing
- h) Knowledge Representation
- i) Expert System
- j) Neural Network

- k) Learning
- l) Planning
- m) Semantic Network

2) Software Components

- a) Machine Language
- b) Assembly language
- c) High level Language
- d) LISP Language
- e) Fourth generation Language
- f) Object Oriented Language
- g) Distributed Language
- h) Natural Language
- i) Particular Problem Solving Language

3) Architectural Components

- a) Uniprocessor
- b) Multiprocessor
- c) Special Purpose Processor
- d) Array Processor
- e) Vector Processor
- f) Parallel Processor
- g) Distributed Processor

10 Definition of Artificial intelligence

1. AI is the study of how to make computers do things which at the moment people do better. This is ephemeral as it refers to the current state of computer science and it excludes a major area ; problems that cannot be solved well either by computers or by people at the moment.
2. AI is a field of study that encompasses computational techniques for performing tasks that apparently require intelligence when performed by humans.
3. AI is the branch of computer science that is concerned with the automation of intelligent behaviour. A I is based upon the principles of computer science namely data structures used in knowledge representation, the algorithms needed to apply that knowledge and the languages and programming techniques used in their implementation.

4. AI is the field of study that seeks to explain and emulate intelligent behaviour in terms of computational processes.
5. AI is about generating representations and procedures that automatically or autonomously solve problems heretofore solved by humans.
6. AI is the part of computer science concerned with designing intelligent computer systems, that is, computer systems that exhibit the characteristics we associate with intelligence in human behaviour such as understanding language, learning, reasoning and solving problems.
7. AI is the study of mental faculties through the use of computational models.
8. AI is the study of the computations that make it possible to perceive, reason, and act.
9. AI is the exciting new effort to make computers think *machines with minds*, in the full and literal sense.
10. AI is concerned with developing computer systems that can store knowledge and effectively use the knowledge to help solve problems and accomplish tasks. This brief statement sounds a lot like one of the commonly accepted goals in the education of humans. We want students to learn (gain knowledge) and to learn to use this knowledge to help solve problems and accomplish tasks.

WEAK AND STRONG AI

There are two conceptual thoughts about AI namely the Weak AI and Strong AI. The strong AI is very much promising about the fact that the machine is almost capable of solve a complex problem like an intelligent man. They claim that a computer is much more efficient to solve the problems than some of the human experts. According to strong AI, the computer is not merely a tool in the study of mind, rather the appropriately programmed computer is really a mind. Strong AI is the supposition that some forms of artificial intelligence can truly reason and solve problems. The term strong AI was originally coined by John Searle.

In contrast, the weak AI is not so enthusiastic about the outcomes of AI and it simply says that some thinking like features can be added to computers to make them more useful tools. It says that computers to make them more useful tools. It says that computers cannot be made intelligent equal to human being, unless constructed significantly differently. They claim that computers may be similar to human experts but not equal in any cases. Generally weak AI refers to the use of software to study or accomplish specific problem solving that do not encompass the full range of human cognitive abilities. An example of weak AI would be a chess program. Weak AI programs cannot be called "intelligent" because they cannot really think.

TASK DOMAIN OF AI

Areas of Artificial Intelligence

- Perception

- **Machine Vision:** It is easy to interface a TV camera to a computer and get an image into memory; the problem is *understanding* what the image represents. Vision takes *lots* of computation; in humans, roughly 10% of all calories consumed are burned in vision computation.
- **Speech Understanding:** Speech understanding is available now. Some systems must be trained for the individual user and require pauses between words. Understanding continuous speech with a larger vocabulary is harder.
- **Touch (*tactile* or *haptic*) Sensation:** Important for robot assembly tasks.

- **Robotics** Although industrial robots have been expensive, robot hardware can be cheap: Radio Shack has sold a working robot arm and hand for \$15. The limiting factor in application of robotics is not the cost of the robot hardware itself. What is needed is perception and intelligence to tell the robot what to do; "blind" robots are limited to very well-structured tasks (like spray painting car bodies).

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

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

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

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

Examples:

- Automatic construction of compiler code generators from a description of a CPU's instruction set.
- J Moore and colleagues proved correctness of the floating-point division algorithm on AMD CPU chip.

- **Symbolic Mathematics** Symbolic mathematics refers to manipulation of *formulas*, rather than arithmetic on numeric values.

- Algebra
- Differential and Integral Calculus

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

- **Game Playing** Games are good vehicles for research because they are well formalized, small, and self-contained. They are therefore easily programmed. Games can be good models of competitive situations, so principles discovered in game-playing programs may be applicable to practical problems.

AI Technique

Intelligence requires knowledge but knowledge possesses less desirable properties such as

- It is voluminous
- it is difficult to characterise accurately
- it is constantly changing
- it differs from data by being organised in a way that corresponds to its application

An AI technique is a method that exploits knowledge that is represented so that

- The knowledge captures generalisations; situations that share properties, are grouped together, rather than being allowed separate representation.
- It can be understood by people who must provide it; although for many programs the bulk of the data may come automatically, such as from readings. In many AI domains people must supply the knowledge to programs in a form the people understand and in a form that is acceptable to the program.
- It can be easily modified to correct errors and reflect changes in real conditions.
- It can be widely used even if it is incomplete or inaccurate.
- It can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities that must be usually considered.

Problem Spaces and Search

Building a system to solve a problem requires the following steps

- Define the problem precisely including detailed specifications and what constitutes an acceptable solution;
- Analyse the problem thoroughly for some features may have a dominant affect on the chosen method of solution;
- Isolate and represent the background knowledge needed in the solution of the problem;
- Choose the best problem solving techniques in the solution.

Defining the Problem as state Search

To understand what exactly artificial intelligence is, we illustrate some common problems. Problems dealt with in artificial intelligence generally use a common term called 'state'. A state represents a status of the solution at a given step of the problem solving procedure. The solution of a problem, thus, is a collection of the problem states. The problem solving procedure applies an operator to a state to get the next state. Then it applies another operator to the resulting state to derive a new state. The process of applying an operator to a state and its subsequent transition to the next state, thus, is continued until the goal (desired) state is derived. Such a method of solving a problem is generally referred to as state space approach. For example, in order to solve the problem play a game, which is restricted to two person table or board games, we require the rules of the game and the targets for winning as well as a means of representing positions in the game. The opening position can be defined as the initial state and a winning position as a goal state, there can be more than one. legal moves allow for transfer from initial state to other states leading to the goal state. However the rules are far too copious in most games especially chess where they exceed the number of particles in the universe 10^{10} . Thus the rules cannot in general be supplied accurately and computer programs cannot easily handle them. The storage also presents another problem but searching can be achieved by hashing. The number of rules that are used must be minimised and the set can be produced by expressing each rule in as general a form as possible. The representation of games in this way leads to a state space representation and it is natural for well organised games with some structure. This representation allows for the formal definition of a problem which necessitates the movement from a set of initial positions to one of a set of target positions. It means that the solution involves using known techniques and a systematic search. This is quite a common method in AI.

Formal description of a problem

- Define a state space that contains all possible configurations of the relevant objects, without enumerating all the states in it. A *state space* represents a problem in terms of *states* and *operators* that change states

- Define some of these states as possible initial states;
- Specify one or more as acceptable solutions, these are goal states;
- Specify a set of rules as the possible actions allowed. This involves thinking about the generality of the rules, the assumptions made in the informal presentation and how much work can be anticipated by inclusion in the rules.

The control strategy is again not fully discussed but the AI program needs a structure to facilitate the search which is a characteristic of this type of program.

Example:

The water jug problem :There are two jugs called **four** and **three** ; four holds a maximum of four gallons and **three** a maximum of three gallons. How can we get 2 gallons in the jug **four**. The state space is a set of ordered pairs giving the number of gallons in the pair of jugs at any time ie (**four**, **three**) where **four** = 0, 1, 2, 3, 4 and **three** = 0, 1, 2, 3. The start state is (0,0) and the goal state is (2,n) where n is a don't care but is limited to **three** holding from 0 to 3 gallons. The major production rules for solving this problem are shown below:

Initial	condition	goal	comment
1 (four,three)	if four < 4	(4,three)	fill four from tap
2 (four,three)	if three < 3	(four,3)	fill three from tap
3 (four,three)	If four > 0	(0,three)	empty four into drain
4 (four,three)	if three > 0	(four,0)	empty three into drain
5 (four,three)	if four+three < 4	(four+three,0)	empty three into four
6 (four,three)	if four+three < 3	(0,four+three)	empty four into three
7 (0,three)	If three > 0	(three,0)	empty three into four
8 (four,0)	if four > 0	(0,four)	empty four into three
9 (0,2)		(2,0)	empty three into four
10 (2,0)		(0,2)	empty four into three
11 (four,three)	if four < 4	(4,three-diff)	pour diff, 4-four, into four from three

12 (three,four) if three<3 (four-diff,3)

pour diff, 3-three, into three from four and

a solution is given below Jug four, jug three rule applied

0 0

0 3 2

3 0 7

3 3 2

4 2 11

0 2 3

2 0 10

Control strategies.

A good control strategy should have the following requirement: The first requirement is that it causes motion. In a game playing program the pieces move on the board and in the water jug problem water is used to fill jugs. The second requirement is that it is systematic, this is a clear requirement for it would not be sensible to fill a jug and empty it repeatedly nor in a game would it be advisable to move a piece round and round the board in a cyclic way. We shall initially consider two systematic approaches to searching.

Monotonic and Non monotonic Learning :

Monotonic learning is when an agent may not learn any knowledge that contradicts what it already knows. For example, it may not replace a statement with its negation. Thus, the knowledge base may only grow with new facts in a monotonic fashion. The advantages of monotonic learning are:

- 1.greatly simplified truth-maintenance
- 2.greater choice in learning strategies

Non-monotonic learning is when an agent may learn knowledge that contradicts what it already knows. So it may replace old knowledge with new if it believes there is sufficient reason to do so. The advantages of non-monotonic learning are:

1.increased applicability to real domains,

2.greater freedom in the order things are learned in

A related property is the consistency of the knowledge. If an architecture must maintain a consistent knowledge base then any learning strategy it uses must be monotonic.

7- PROBLEM CHARACTERISTICS

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

- ⇒ Is the problem decomposable into set of sub problems?
- ⇒ Can the solution step be ignored or undone?
- ⇒ Is the problem universally predictable?
- ⇒ Is a good solution to the problem obvious without comparison to all the possible solutions?
- ⇒ Is the desire solution a state of world or a path to a state?
- ⇒ Is a large amount of knowledge absolutely required to solve the problem?
- ⇒ Will the solution of the problem required interaction between the computer and the person?

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

PRODUCTION SYSTEM AND ITS CHARACTERISTICS

The production system is a model of computation that can be applied to implement search algorithms and model human problem solving. Such problem solving knowledge can be packed up in the form of little quanta called productions. A production is a rule consisting of a situation recognition part and an action part. A production is a situation-action pair in which the left side is a list of things to watch for and the right side is a list of things to do so. When productions are used in deductive systems, the situation that trigger productions are specified combination of facts. The actions are restricted to being assertion of new facts deduced directly from the triggering combination. Production systems may be called premise conclusion pairs rather than situation action pair.

A production system consists of following components.

- (a) A set of production rules, which are of the form $A \rightarrow B$. Each rule consists of left hand side constituent that represent the current problem state and a right hand side that represent an output state. A rule is applicable if its left hand side matches with the current problem state.

- (b) A database, which contains all the appropriate information for the particular task. Some part of the database may be permanent while some part of this may pertain only to the solution of the current problem.
- (c) A control strategy that specifies order in which the rules will be compared to the database of rules and a way of resolving the conflicts that arise when several rules match simultaneously.
- (d) A rule applier, which checks the capability of rule by matching the content state with the left hand side of the rule and finds the appropriate rule from database of rules.

The important roles played by production systems include a powerful knowledge representation scheme. A production system not only represents knowledge but also action. It acts as a bridge between AI and expert systems. Production system provides a language in which the representation of expert knowledge is very natural. We can represent knowledge in a production system as a set of rules of the form

If (condition) THEN (condition)

along with a control system and a database. The control system serves as a rule interpreter and sequencer. The database acts as a context buffer, which records the conditions evaluated by the rules and information on which the rules act. The production rules are also known as condition – action, antecedent – consequent, pattern – action, situation – response, feedback – result pairs.

For example,

If (you have an exam tomorrow)

THEN (study the whole night)

The production system can be classified as monotonic, non-monotonic, partially commutative and commutative.

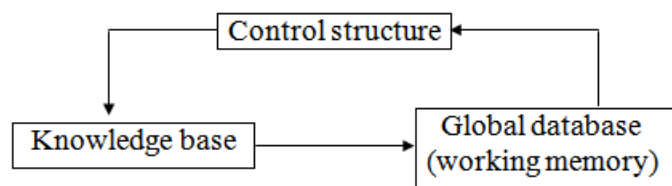


Figure Architecture of Production System

Features of Production System

Some of the main features of production system are:

Expressiveness and intuitiveness: In real world, many times situation comes like “if this happen-you will do that”, “if this is so-then this should happen” and many more. The production rules essentially tell us what to do in a given situation.

1. **Simplicity:** The structure of each sentence in a production system is unique and uniform as they use “IF-THEN” structure. This structure provides simplicity in knowledge representation. This feature of production system improves the readability of production rules.
2. **Modularity:** This means production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no deleterious side effects.
3. **Modifiability:** This means the facility of modifying rules. It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.
4. **Knowledge intensive:** The knowledge base of production system stores pure knowledge. This part does not contain any type of control or programming information. Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation.

Disadvantages of production system

1. **Opacity:** This problem is generated by the combination of production rules. The opacity is generated because of less prioritization of rules. More priority to a rule has the less opacity.
2. **Inefficiency:** During execution of a program several rules may active. A well devised control strategy reduces this problem. As the rules of the production system are large in number and they are hardly written in hierarchical manner, it requires some forms of complex search through all the production rules for each cycle of control program.
3. **Absence of learning:** Rule based production systems do not store the result of the problem for future use. Hence, it does not exhibit any type of learning capabilities. So for each time for a particular problem, some new solutions may come.
4. **Conflict resolution:** The rules in a production system should not have any type of conflict operations. When a new rule is added to a database, it should ensure that it does not have any conflicts with the existing rules.

ALGORITHM OF PROBLEM SOLVING

Any one algorithm for a particular problem is not applicable over all types of problems in a variety of situations. So there should be a general problem solving algorithm, which may work for different strategies of different problems.

Algorithm (problem name and specification)

Step 1:

Analyze the problem to get the starting state and goal state.

Step 2:

Find out the data about the starting state, goalstate

Step 3:

Find out the production rules from initial database for proceeding the problem to goal state.

Step 4:

Select some rules from the set of rules that can be applied to data.

Step 5:

Apply those rules to the initial state and proceed to get the next state.

Step 6:

Determine some new generated states after applying the rules. Accordingly make them as current state.

Step 7:

Finally, achieve some information about the goal state from the recently used current state and get the goal state.

Step 8:

Exit.

After applying the above rules an user may get the solution of the problem from a given state to another state. Let us take few examples.

VARIOUS TYPES OF PROBLEMS AND THEIR SOLUTIONS

Water Jug Problem

Definition:

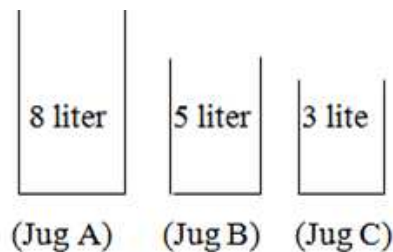
Some jugs are given which should have non-calibrated properties. At least any one of the jugs should have filled with water. Then the process through which we can divide the whole water into different jugs according to the question can be called as water jug problem.

Procedure:

Suppose that you are given 3 jugs A,B,C with capacities 8,5 and 3 liters respectively but are not calibrated (i.e. no measuring mark will be there). Jug A is filled with 8 liters of water. By a series of pouring back and forth among the 3 jugs, divide the 8 liters into 2 equal parts i.e. 4 liters in jug A and 4 liters in jug B. How?

In this problem, the start state is that the jug A will contain 8 liters water whereas jug B and jug C will be empty. The production rules involve filling a jug with some amount of water, taking from the jug A. The search will be finding the sequence of production rules which transform the initial state to final state. The state space for this problem can be described by set of ordered pairs of three variables (A, B, C) where variable A represents the 8 liter jug, variable B represents the 5 liter and variable C represents the 3 liters jug respectively.

Figure



The production rules are formulated as follows:

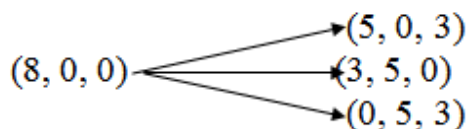
Step 1:

In this step, the initial state will be (8, 0, 0) as the jug B and jug C will be empty. So the water of jug A can be poured like:

(5, 0, 3) means 3 liters to jug C and 5 liters will remain in jug A.

(3, 5, 0) means 5 liters to jug B and 3 liters will be in jug A.

(0, 5, 3) means 5 liters to jug B and 3 liters to jug C and jug C and jug A will be empty.



Step2:

In this step, start with the first current state of step-1 i.e. (5, 0, 3). This state can only be implemented by pouring the 3 liters water of jug C into jug B. so the state will be (5, 3, 0). Next, come to the second

current state of step-1 i.e. (3, 5, 0). This state can be implemented by only pouring the 5 liters water of jug B into jug C. So the remaining water in jug B will be 2 liters. So the state will be (3, 2, 3). Finally come to the third current state of step-1 i.e. (0, 5, 3). But from this state no more state can be implemented because after implementing we may get (5, 0, 3) or (3, 5, 0) or (8, 0, 0) which are repeated state. Hence these states are not considerably again for going towards goal.

So the state will be like:

$$(5, 0, 3) \rightarrow (5, 3, 0)$$

$$(3, 5, 0) \rightarrow (3, 2, 3)$$

$$(0, 5, 3) \rightarrow X$$

Step 3:

In this step, start with the first current state of step-2 i.e. (5, 3, 0) and proceed likewise the above steps.

$$(5, 3, 0) \rightarrow (2, 3, 3)$$

$$(3, 2, 3) \rightarrow (6, 2, 0)$$

Step 4:

In this step, start with the first current state of step-3 i.e. (2, 3, 3) and proceed.

$$(2, 3, 3) \rightarrow (2, 5, 1)$$

$$(6, 2, 0) \rightarrow (7, 0, 1)$$

Step 5:

$$(2, 5, 1) \rightarrow (7, 0, 1)$$

$$(6, 0, 2) \rightarrow (1, 5, 2)$$

Step6:

$$(7, 0, 1) \rightarrow (7, 1, 0)$$

$$(1, 4, 3) \rightarrow (1, 4, 3)$$

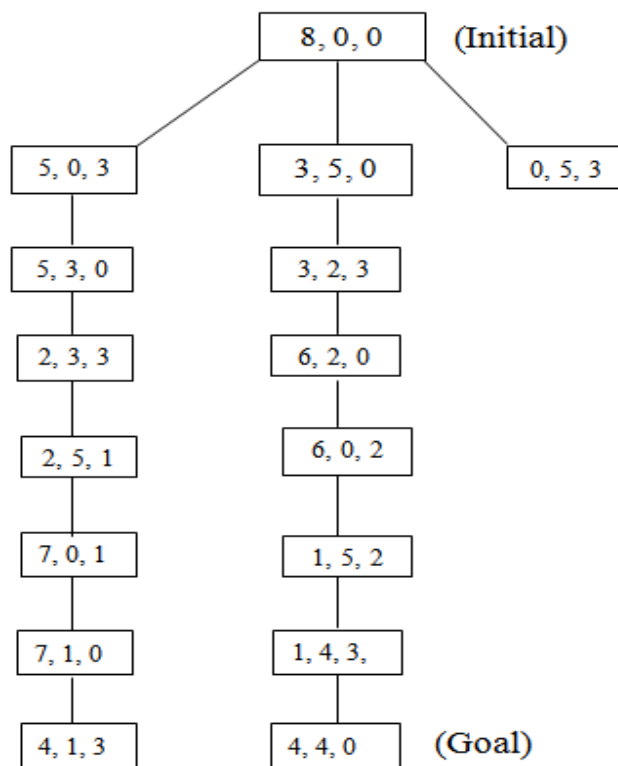
Step7:

$$(7, 1, 0) \rightarrow (4, 1, 3)$$

$(1, 4, 3) \rightarrow (4, 4, 0)$ (Goal)

So finally the state will be $(4, 4, 0)$ that means jug A and jug B contains 4 liters of water each which is our goal state. One thing you have to very careful about the pouring of water from one jug to another that the capacity of jug must satisfy the condition to contain that much of water.

The tree of the water jug problem can be like:



Figure

Comments:

- ⇒ This problem takes a lot of time to find the goal state.
- ⇒ This process of searching in this problem is very lengthy.

⇒ At each step of the problem the user have to strictly follow the production rules. Otherwise the problem may go to infinity step.

Missionaries and Carnivals Problem

Definition:

In Missionaries and Carnivals Problem, initially there are some missionaries and some carnivals will be at a sideof a river. They want to cross the river. But there is only one boat available to cross the river. The capacity of the boat is 2 and no one missionary or no Carnivals can cross the river together. So for solving the problem and to find out the solution on different states is called the Missionaries and Carnival Problem.

Procedure:

Let us take an example. Initially a boatman, Grass, Tiger and Goat is present at the left bank of the river and want to cross it. The only boat available is one capable of carrying 2 objects of portions at a time. The condition of safe crossing is that at no time the tiger present with goat, the goat present with the grass at the either side of the river. How they will cross the river?

The objective of the solution is to find the sequence of their transfer from one bank of the river to the other using the boat sailing through the river satisfying these constraints.

Let us use different representations for each of the missionaries and Carnivals as follows.

B: Boat

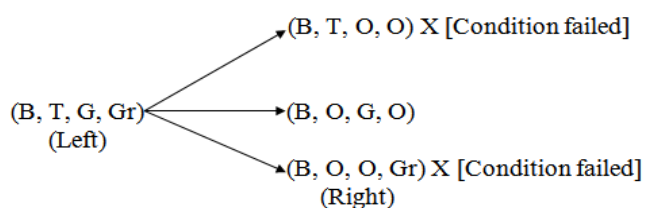
T: Tiger

G: Goat

Gr: Grass

Step 1:

According to the question, this step will be (B, T, G, Gr) as all the Missionaries and the Carnivals are at one side of the bank of the river. Different states from this state can be implemented as



The states (B, T, O, O) and (B, O, O, Gr) will not be countable because at a time the Boatman and the Tiger or the Boatman and grass cannot go. (According to the question).

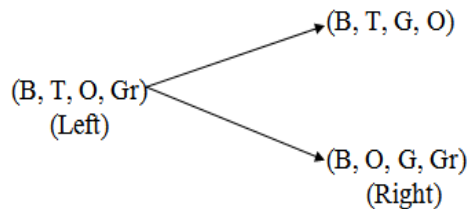
Step 2:

Now consider the current state of step-1 i.e. the state (B, O, G, O). The state is the right side of the river. So on the left side the state may be (B, T, O, Gr)

i.e. (B, O, G, O) → (B, T, O, Gr)
 (Right) (Left)

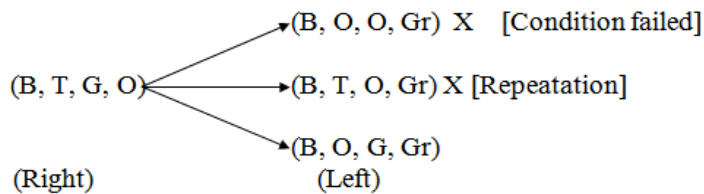
Step 3:

Now proceed according to the left and right sides of the river such that the condition of the problem must be satisfied.

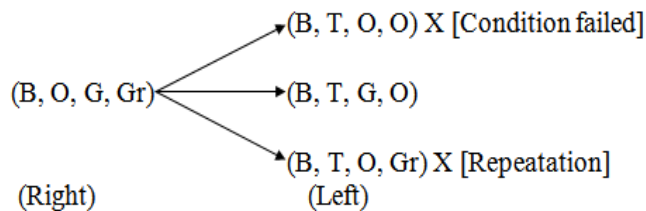


Step 4:

First, consider the first current state on the right side of step 3 i.e.

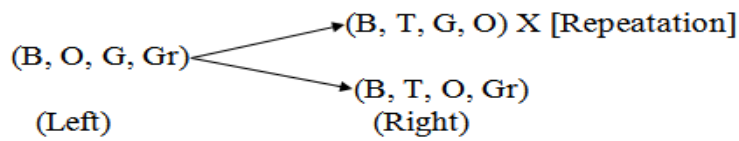


Now consider the second current state on the right side of step-3 i.e.

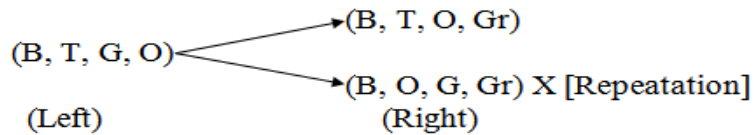


Step 5:

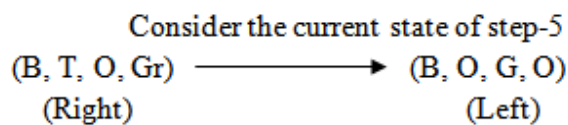
Now first consider the first current state of step 4 i.e.



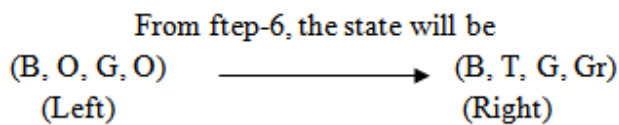
Now consider the 2nd state of step-4 i.e.



Step 6:



Step 7:



Hence the final state will be (B, T, G, Gr) which are on the right side of the river.

Comments:

- ⇒ This problem requires a lot of space for its state implementation.
- ⇒ It takes a lot of time to search the goal node.
- ⇒ The production rules at each level of state are very strict.

Chess Problem

Definition:

It is a normal chess game. In a chess problem, the start is the initial configuration of chessboard. The final state is the any board configuration, which is a winning position for any player. There may be multiple final positions and each board configuration can be thought of as representing a state of the game. Whenever any player moves any piece, it leads to different state of game.

Procedure:

1	2	3
4	5	6
7	8	9

A 3x3 Chess board

Figure

The above figure shows a 3x3 chessboard with each square labeled with integers 1 to 9. We simply enumerate the alternative moves rather than developing a general move operator because of the reduced size of the problem. Using a predicate called `move` in predicate calculus, whose parameters are the starting and ending squares, we have described the legal moves on the board. For example, `move(1, 8)` takes the knight from the upper left-hand corner to the middle of the bottom row. While playing Chess, a knight can move two squares either horizontally or vertically followed by one square in an orthogonal direction as long as it does not move off the board.

The all possible moves of figure are as follows.

- Move (1, 8) move (6, 1)
- Move (1, 6) move (6, 7)
- Move (2, 9) move (7, 2)
- Move (2, 7) move (7, 6)
- Move (3, 4) move (8, 3)
- Move (3, 8) move (8, 1)
- Move (4, 1) move (9, 2)
- Move (4, 3) move (9, 4)

The above predicates of the Chess Problem form the knowledge base for this problem. An unification algorithm is used to access the knowledge base.

Suppose we need to find the positions to which the knight can move from a particular location, square 2. The goal `move(z, x)` unifies with two different predicates in the knowledge base, with the substitutions $\{7/x\}$ and $\{9/x\}$. Given the goal `move(2, 3)`, the responsible is failure, because no `move(2, 3)` exists in the knowledge base.

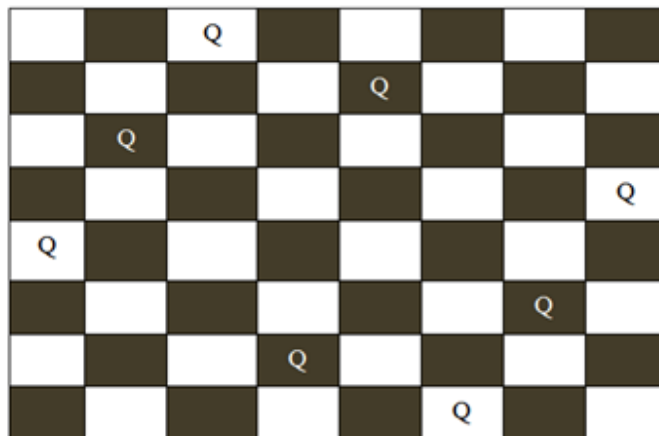
Comments:

- ⇒ In this game a lots of production rules are applied for each move of the square on the chessboard.
- ⇒ A lots of searching are required in this game.
- ⇒ Implementation of algorithm in the knowledge base is very important.

8- Queen Problem

Definition:

“We have 8 queens and an 8x8 Chess board having alternate black and white squares. The queens are placed on the chessboard. Any queen can attack any other queen placed on same row, or column or diagonal. We have to find the proper placement of queens on the Chess board in such a way that no queen attacks other queen”.



Procedure:

Figure A possible board configuration of 8 queen problem

In figure , the possible board configuration for 8-queen problem has been shown. The board has alternative black and white positions on it. The different positions on the board hold the queens. The production rule for this game is you cannot put the same queens in a same row or same column or in same diagonal. After shifting a single queen from its position on the board, the user have to shift other queens according to the production rule. Starting from the first row on the board the queen of their corresponding row and column are to be moved from their original positions to another position. Finally the player has to be ensured that no rows or columns or diagonals of on the table is same.

Comments:

- ⇒ This problem requires a lot of space to store the board.

- ⇒ It requires a lot of searching to reach at the goal state.
- ⇒ The time factor for each queen's move is very lengthy.
- ⇒ The problem is very strict towards the production rules.

8- Puzzle Problem

Definition:

“It has set off a 3x3 board having 9 block spaces out of which 8 blocks having tiles bearing number from 1 to 8. One space is left blank. The tile adjacent to blank space can move into it. We have to arrange the tiles in a sequence for getting the goal state”.

Procedure:

The 8-puzzle problem belongs to the category of “sliding block puzzle” type of problem. The 8-puzzle is a square tray in which eight square tiles are placed. The remaining ninth square is uncovered. Each tile in the tray has a number on it. A tile that is adjacent to blank space can be slide into that space. The game consists of a starting position and a specified goal position. The goal is to transform the starting position into the goal position by sliding the tiles around. The control mechanisms for an 8-puzzle solver must keep track of the order in which operations are performed, so that the operations can be undone one at a time if necessary. The objective of the puzzles is to find a sequence of tile movements that leads from a starting configuration to a goal configuration such as two situations given below.

Figure (Starting State)

(Goal State)

The state of 8-puzzle is the different permutation of tiles within the frame. The operations are the permissible moves up, down, left, right. Here at each step of the problem a function $f(x)$ will be defined

1	2	3
4	5	6
	7	8

1	2	3
4	5	6
7	8	-

which is the combination of $g(x)$ and $h(x)$.

i.e.

$$F(x) = g(x) + h(x)$$

Where

$g(x)$: how many steps in the problem you have already done or the current state from the initial state.

$h(x)$: Number of ways through which you can reach at the goal state from the current state or

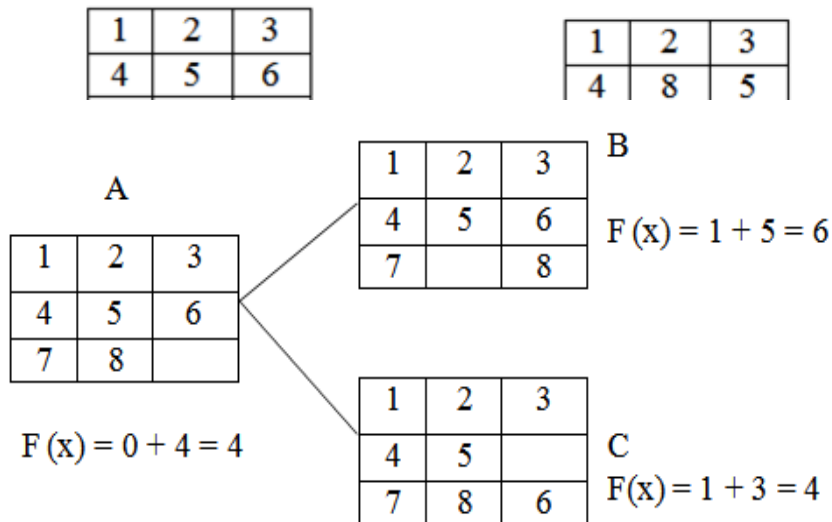
Or

$h(x)$ is the heuristic estimator that compares the current state with the goal state note down how many states are displaced from the initial or the current state. After calculating the $f(x)$ value at each step finally take the smallest $f(x)$ value at every step and choose that as the next current state to get the goal state.

Let us take an example.

Figure (Initial State)

(Goal State)

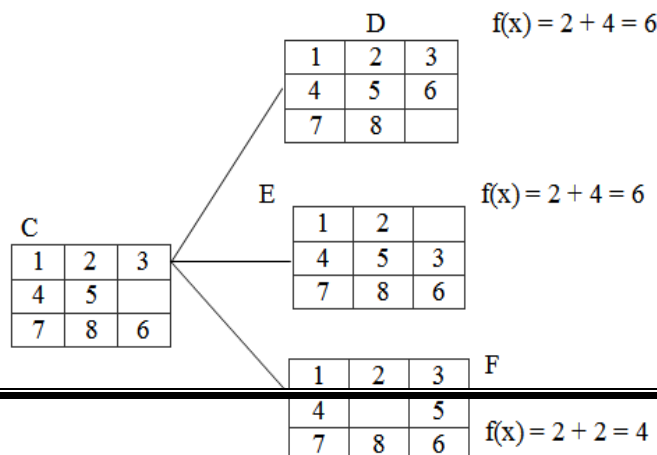


Step1:

$f(x)$ is the step required to reach at the goal state from the initial state. So in the tray either 6 or 8 can change their portions to fill the empty position. So there will be two possible current states namely B and C. The $f(x)$ value of B is 6 and that of C is 4. As 4 is the minimum, so take C as the current state to the next state.

Step 2:

In this step, from the tray C three states can be drawn. The empty position will contain either 5 or 3 or 6. So for three different values three different states can be obtained. Then calculate each of their $f(x)$ and

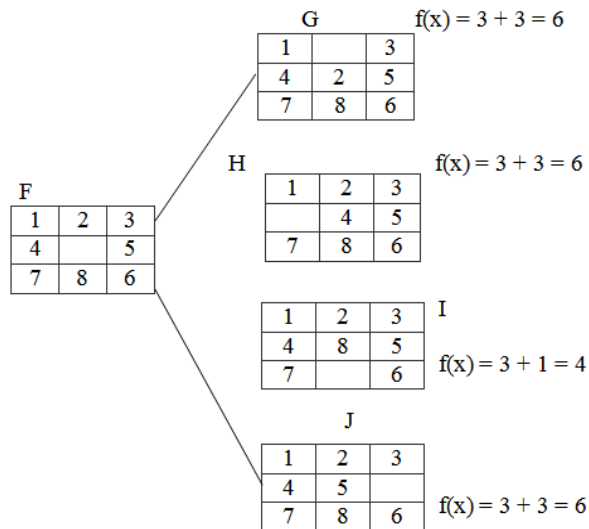


take the minimum one.

Here the state F has the minimum value i.e. 4 and hence take that as the next current state.

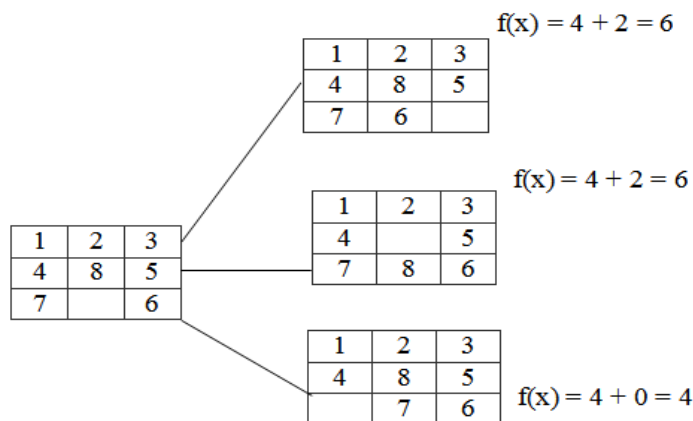
Step 3:

The tray F can have 4 different states as the empty positions can be filled with b4 values i.e.2, 4, 5, 8.



Step 4:

In the step-3 the tray I has the smallest f (n) value. The tray I can be implemented in 3 different states because the empty position can be filled by the members like 7, 8, 6.



Hence, we reached at the goal state after few changes of tiles in different positions of the trays.

Comments:

- ⇒ This problem requires a lot of space for saving the different trays.
- ⇒ Time complexity is more than that of other problems.

- ⇒ The user has to be very careful about the shifting of tiles in the trays.
- ⇒ Very complex puzzle games can be solved by this technique.

Monkey Banana Problem

Definition:

“A monkey is in a room. A bunch of bananas is hanging from the ceiling. The monkey cannot reach the bananas directly. There is a box in the corner of the room. How can the monkey get the bananas?”

Procedure:

The solution of the problem is of course that the monkey must push the box under the bananas, then stand on the box and grab the bananas. But the solution procedure requires a lot of planning algorithms. The purpose of the problem is to raise the question: Are monkeys intelligent? Both humans and monkeys have the ability to use mental maps to remember things like where to go to find shelter or how to avoid danger. They can also remember where to go to gather food and water, as well as how to communicate with each other. Monkeys have the ability not only to remember how to hunt and gather but they also have the ability to learn new things, as is the case with the monkey and the bananas. Even though that monkey may never have entered that room before or had only a box for a tool to gather the food available, that monkey can learn that it needs to move the box across the floor, position it below the bananas and climb the box to reach for them. Some people believe that this is part instinct, part learned behaviour. It is most probably both.

Initially, the monkey is at location ‘A’, the banana is at location ‘B’ and the box is at location ‘C’. The monkey and box have height “low”; but if the monkey climbs onto the box will have height “High”, the same as the bananas.

The action available to the monkey include:

“GO” from one place to another.

“PUSH” an object from one place to another.

“Climb” onto an object.

“Grasp” an object.

Grasping results in holding the object if the monkey and the object are in the same place at the same height.

The solution of the problem in different steps can be of followings.

1. What is the initial state description?
 At (monkey, A), At (banana, B), At (box, C)
 Position (monkey, low), Position (banana, high), Position (box, low)

2. What are the definitions of the different actions?

a) Go (x, y)
 Precondition: At (monkey, x)
 Effects: \neg At (monkey, x), At (monkey, y)

b) Push (object, x, y, height)
 Pre condition: At (monkey, x), At (object, x), Position (monkey, height), Position (object, height)
 Effects: \neg (monkey, x), \neg At (object, x), At (monkey, y), At(object, y)

c) Climb up (object, y)
 Precondition: At (monkey, x), At (object, x), Position (monkey, low), Position (object, low)
 Effects: \neg Position (monkey, low), Position (monkey, high), On (monkey, object)

d) Climb down (object)
 Preconditions: Position (monkey, high), On (monkey, object)
 Effects: \neg Position (monkey, high), \neg On (monkey, object)
 Position: (monkey, low)

e) Grasp (object, x, height)
 Preconditions: At (monkey, x), At (object, x), Position (monkey, height), Position (object, height)
 Effect: Hold (object)

f) UnGrasp (object, x, height)
 Preconditions: Hold (object), At (monkey, x), At (object, x), Position (monkey, height), Position (object, height)
 Effects: \neg Hold (object)

So the solution to the planning problem may be of following

- \Rightarrow GO(A,C)
- \Rightarrow PUSH (Box, C, B, Low)
- \Rightarrow Climb Up(Box , B)
- \Rightarrow Grasp(banana, B, High)
- \Rightarrow Climb down(Box)
- \Rightarrow Push(Box, B, C, Low)

Comments:

\Rightarrow One major application of the monkey banana problem is the toy problem of computer science.

- ⇒ One of the specialized purposes of the problem is to raise the question: Are monkeys intelligent?
- ⇒ This problem is very useful in logic programming and planning.

Tower of Hanoi Problem

Definition:

“We are given a tower of eight discs (initially) four in the applet below, initially stacked in increasing size on one of three pegs. The objective is to transfer the entire tower to one of the other pegs (the right most one in the applet below), moving only one disc at a time and never a larger one onto a smaller”.

Procedure:

The tower of Hanoi puzzle was invented by the French mathematician Eduardo Lucas in 1883. The puzzle is well known to students of computer science since it appears in virtually any introductory text on data structure and algorithms.

The objective of the puzzle is to move the entire stack to another rod, obeying the following rules.

- ⇒ Only one disc can be moved at a time.
- ⇒ Each move consist of taking the upper disc from one of the rods and sliding it onto another rod, on top of the other discs that may already be present on that rod.
- ⇒ No disc may be placed on the top of a smaller disk.

There is a legend about a Vietnamese temple which contains a large room with three times. Worn posts in it surrounded by 64 golden disks. The priests of Hanoi, acting out of command of an ancient prophecy, have been moving these disks, in accordance with the rules of the puzzle, since that time. The puzzle is therefore also known as the tower of Brahma puzzle. According to the legend, when the last move of the puzzle is completed, the world will end.

There are many variations on this legend. For instance, in some tellings, the temple is a monastery and the priests are monks. The temple or monastery may be said to be in different parts of the world including Hanoi, Vietnam and may be associated with any religion. The flag tower of Hanoi may have served as the inspiration for the name.

The puzzle can be played with any number of disks, although many toy versions have around seven to nine of them. The game seems impossible to many novices yet is solvable with a simple algorithm. The following solution is a very simple method to solve the tower of Hanoi problem.

- ⇒ Alternative moves between the smallest piece and a non- smallest piece. When moving the smallest piece, always move it in the same direction (to the right if starting number of pieces is even, to the left if starting number of pieces is odd).

- ⇒ If there is no tower in the chosen direction, move the pieces to the opposite end, but then continue to move in the correct direction, for example if you started with three pieces, you would move the smallest piece to the opposite end, then continue in the left direction after that.

- ⇒ When the turn is to move the non-smallest piece, there is only one legal move.

Doing this should complete the puzzle using the least amount of moves to do so. Finally, the user will reach at the goal. Also various types of solutions may be possible to solve the tower of Hanoi problem like recursive procedure, non-recursive procedure and binary solution procedure.

Another simple solution to the problem is given below.

For an even number of disks

- ⇒ Make the legal move between pegs A and B.
- ⇒ Make the legal moves between pages A and C.
- ⇒ Make the legal move between pages B and C.

For an even number of disks

- ⇒ Make the legal move between pegs A and C.
- ⇒ Make the legal move between pegs A and B.
- ⇒ Make the legal move between pegs B and C.
- ⇒ Repeat until complete.

A recursive solution for tower of Hanoi problem is as follows.

A key to solving this problem is to recognize that it can be solve by breaking the problem down into the collection of smaller problems and further breaking those problems down into even smaller problems until a solution is reached. The following procedure demonstrates this approach.

- ⇒ Label the pegs A, B, C - these levels may move at different steps.
- ⇒ Let n be the total number of disks.
- ⇒ Number of disks from 1 (smallest, topmost) to n (largest, bottommost).

To move n disks from peg A to peg C.

- a) Move $n-1$ disks from A to B. This leaves disk $\#n$ alone on peg A.
- b) Move disk $\#n$ from A to C.
- c) Move $n-1$ disks from B to C so they sit on disk $\#n$.

To carry out steps a and c, apply the same algorithm again for $n-1$. The entire procedure is a finite number of steps, since at most point the algorithm will be required for $n = 1$. This step, moving a single disc from peg A to peg B, is trivial.

Comments:

- ⇒ The tower of Hanoi is frequently used in psychological research on problem solving.
- ⇒ This problem is frequently used in neuro-psychological diagnosis and treatment of executive functions.
- ⇒ The tower of Hanoi is also used as backup rotation scheme when performing computer data backups where multiple tabs/media are involved.
- ⇒ This problem is very popular for teaching recursive algorithm to beginning programming students.
- ⇒ A pictorial version of this puzzle is programmed into emacs editor, accessed by typing M - X Hanoi.
- ⇒ The tower of Hanoi is also used as a test by neuro-psychologists trying to evaluate frontal lobe deficits.

Cryptarithmic Problem

Definition:

“It is an arithmetic problem which is represented in letters. It involves the decoding of digit represented by a character. It is in the form of some arithmetic equation where digits are distinctly represented by some characters. The problem requires finding of the digit represented by each character. Assign a decimal digit to each of the letters in such a way that the answer to the problem is correct. If the same letter occurs more than once, it must be assigned the same digit each time. No two different letters may be assigned the same digit”.

Procedure:

Cryptarithmic problem is an interesting constraint satisfaction problem for which different algorithms have been developed. Cryptarithm is a mathematical puzzle in which digits are replaced by letters of the alphabet or other symbols. Cryptarithmic is the science and art of creating and solving cryptarithms.

The different constraints of defining a cryptarithmic problem are as follows.

- 1) Each letter or symbol represented only one and a unique digit throughout the problem.
- 2) When the digits replace letters or symbols, the resultant arithmetical operation must be correct.

The above two constraints lead to some other restrictions in the problem.

For example:

$$\begin{array}{r}
 . \text{EAT} \\
 + \text{THAT} \\
 \hline
 \text{A P P L E}
 \end{array}$$

Consider that, the base of the number is 10. Then there must be at most 10 unique symbols or letters in the problem. Otherwise, it would not possible to assign a unique digit to unique letter or symbol in the problem. To be semantically meaningful, a number must not begin with a 0. So, the letters at the beginning of each number should not correspond to 0. Also one can solve the problem by a simple blind search. But a rule based searching technique can provide the solution in minimum time.

Now, let us solve a simple cryptarithmic puzzle given below.

Step 1:

In the above problem, M must be 1. You can visualize that, this is an addition problem. The sum of two four digit numbers cannot be more than 10,000. Also M cannot be zero according to the rules, since it is

$$\begin{array}{r}
 \text{S E N D} \\
 + \text{M O R E} \\
 \hline
 \text{M O N E Y}
 \end{array}$$

the first letter.

So now you have the problem like

$$\begin{array}{r}
 \text{S E N D} \\
 + 1 0 \text{ R E} \\
 \hline
 1 0 \text{ N Y}
 \end{array}$$

Step 2:

Now in the column s10, $s+1 \geq 10$. S must be 8 because there is a 1 carried over from the column EON or 9. O must be 0 (if $s=8$ and there is a 1 carried or $s = 9$ and there is no 1 carried) or 1 (if $s=9$ and there is a 1 carried). But 1 is already taken, so O must be 0.

$$\begin{array}{r}
 \text{S E N D} \\
 + 10 \text{ R E} \\
 \hline
 10 \text{ N E Y}
 \end{array}$$

Step 3:

There cannot be carry from column EON because any digit $+0 < 10$, unless there is a carry from the column NRE, and $E=9$; But this cannot be the case because then N would be 0 and 0 is already taken. So $E < 9$ and there is no carry from this column. Therefore $S=9$ because $9+1=10$.

Step 4:

$$\begin{array}{r}
 9 \text{ E N D} \\
 + 10 \text{ R E} \\
 \hline
 10 \text{ N E Y}
 \end{array}$$

In the column EON, E cannot be equal to N. So there must be carry from the column NRE; $E+1=N$. We now look at the column NRE, we know that $E+1=N$. Since we know that carry from this column, $N+R=1E$ (if there is no carry from the column DEY) or $N+R+1=1E$ (if there is a carry from the column DEY).

Let us see both the cases:

$$\text{No carry: } N + R = 10 + (N - 1) = N + 9$$

$$R = 9$$

But 9 is already taken, so this will not work

$$\text{Carry: } N + R + 1 = 9$$

$$R = 9 - 1 = 8 \text{ This must be solution for R}$$

$$\begin{array}{r}
 9 \text{ E N D} \\
 + 108 \text{ E} \\
 \hline
 10 \text{ N E Y}
 \end{array}$$

Step 5:

Now just think what are the digits we have left? They are 7, 6, 5, 4, 3 and 2. We know there must be a carry from the column DEY. So $D + E > 10$. $N = E + 1$, So E cannot be 7 because then N would be 8

$$\begin{array}{r}
 95 \text{ N D} \\
 + 1085
 \end{array}$$

$$10 \text{ N 5 Y}$$

which is already taken. D is almost 7, so E cannot be 2 because then $D + E < 10$ and E cannot be 3 because then $D + E = 10$ and $Y = 0$, but 0 is already taken. Also E cannot be 4 because if $D > 6$, $D + E < 10$ and if $D = 6$ or $D = 7$ then $Y = 0$ or $Y = 1$, which are both taken. So E is 5 or 6. If $E = 6$, then $D = 7$ and $Y = 3$. So this part will work but look the column N8E. Point that there is a carry from the column D5Y. $N + 8 + 1 = 16$ (As there is a carry from this column). But then $N=7$ and 7 is taken by D therefore $E=5$.

Step 6:

Now we have gotten this important digit, it gets much simpler from here. $N+8+1=15$, $N=6$

$$\begin{array}{r} 956D \\ + 1085 \\ \hline 1065Y \end{array}$$

Step 7:

The digits left are 7, 4, 3 and 2. We know there is carry from the column D5Y, so the only pair that works is $D=7$ and $Y=2$.

$$\begin{array}{r} 9567 \\ + 1085 \\ \hline 10652 \end{array}$$

Which is final solution of the problem.

Comments:

- ⇒ This problem requires a lot of reasoning.
- ⇒ Time complexity of the problem is more as concerned to the other problems.
- ⇒ This problem can also be solved by the evolutionary approach and mutation operations.
- ⇒ This problem is dependent upon some constraints which are necessary part of the problem.
- ⇒ Various complex problems can also be solved by this technique.

SEARCHING

Problem solving in artificial intelligence may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. In AI problem solving by search

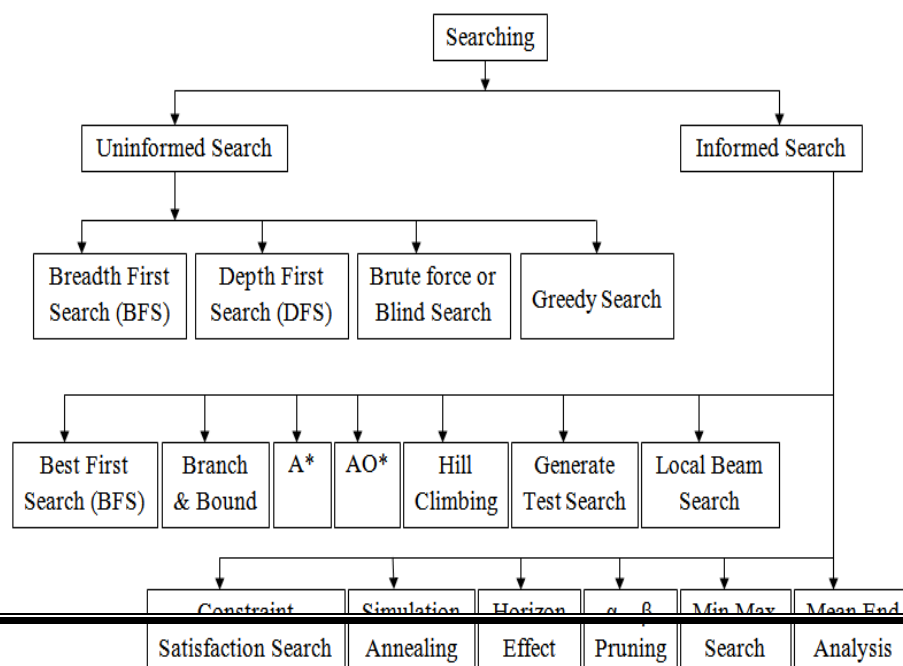
algorithms is quite common technique. In the coming age of AI it will have big impact on the technologies of the robotics and path finding. It is also widely used in travel planning. This chapter contains the different search algorithms of AI used in various applications. Let us look the concepts for visualizing the algorithms.

A search algorithm takes a problem as input and returns the solution in the form of an action sequence. Once the solution is found, the actions it recommends can be carried out. This phase is called as the execution phase. After formulating a goal and problem to solve the agent cells a search procedure to solve it. A problem can be defined by 5 components.

- a) **The initial state:** The state from which agent will start.
- b) **The goal state:** The state to be finally reached.
- c) **The current state:** The state at which the agent is present after starting from the initial state.
- d) **Successor function:** It is the description of possible actions and their outcomes.
- e) **Path cost:** It is a function that assigns a numeric cost to each path.

DIFFERENT TYPES OF SEARCHING

the searching algorithms can be various types. When any type of searching is performed, there may some information about the searching or mayn't be. Also it is possible that the searching procedure may depend upon any constraints or rules. However, generally searching can be classified into two types i.e. uninformed searching and informed searching. Also some other classifications of these searches are given below in the figure .



Figure

UNINFORMED SEARCH

Breadth First Search (BFS)

Breadth first search is a general technique of traversing a graph. Breadth first search may use more memory but will always find the shortest path first. In this type of search the state space is represented in form of a tree. The solution is obtained by traversing through the tree. The nodes of the tree represent the start value or starting state, various intermediate states and the final state. In this search a queue data structure is used and it is level by level traversal. Breadth first search expands nodes in order of their distance from the root. It is a path finding algorithm that is capable of always finding the solution if one exists. The solution which is found is always the optional solution. This task is completed in a very memory intensive manner. Each node in the search tree is expanded in a breadth wise at each level.

Concept:

Step 1: Traverse the root node

Step 2: Traverse all neighbours of root node.

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

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

Algorithm:

Step 1: Place the root node inside the queue.

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

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

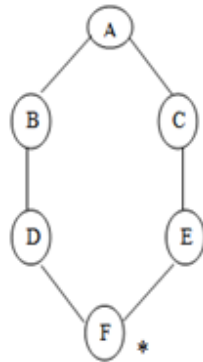
Step 4: Remove the FRONT node from the queue. Process it and find all its neighbours that are in ready state then place them inside the queue in any order.

Step 5: Go to Step 3.

Step 6: Exit.

Implementation:

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

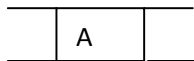


Figure

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

Step 1:

Place the root node inside the queue i.e. A

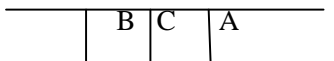


Step 2:

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

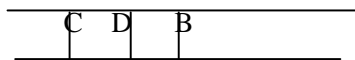
Step 3:

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



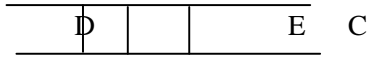
Step 4:

Now B is the FRONT node of the queue. So process B and find the neighbours of B i.e. D.

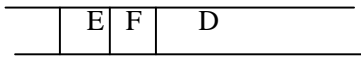


Step 5:

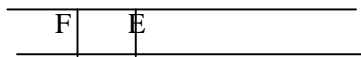
Now find out the neighbours of C i.e. E

**Step 6:**

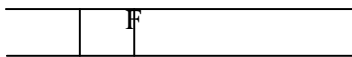
Next find out the neighbours of D as D is the FRONT node of the queue

**Step 7:**

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

**Step 8:**

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

**Advantages:**

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

Disadvantages:

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

Depth First Search (DFS)

DFS is also an important type of uniform search. DFS visits all the vertices in the graph. This type of algorithm always chooses to go deeper into the graph. After DFS visited all the reachable vertices from a particular sources vertices it chooses one of the remaining undiscovered vertices and continues the search. DFS reminds the space limitation of breath first search by always generating next a child of the deepest unexpanded noded. The data structure stack or last in first out (LIFO) is used for DFS. One interesting

property of DFS is that, the discover and finish time of each vertex from a parenthesis structure. If we use one open parenthesis when a vertex is finished then the result is properly nested set of parenthesis.

Concept:

Step 1: Traverse the root node.

Step 2: Traverse any neighbour of the root node.

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

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

Algorithm:

Step 1: PUSH the starting node into the stack.

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

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

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

Step 5: Go to step 3.

Step 6: Exit.

Implementation:

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

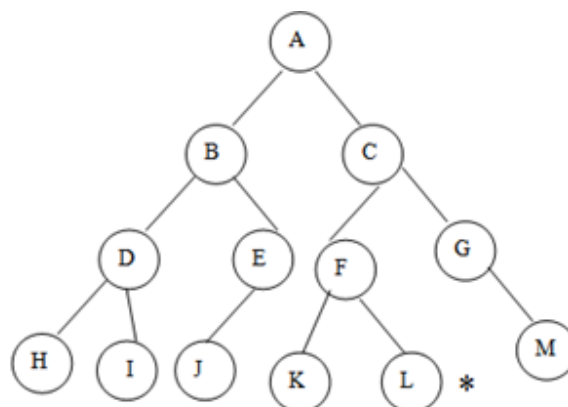
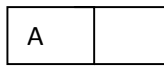


Figure Examples of DFS

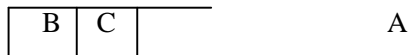
Consider A as the root node and L as the goal node in the graph figure

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

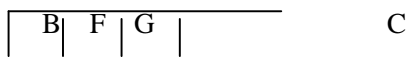


Step 2: Now the stack is not empty and A is not our goal node. Hence move to next step.

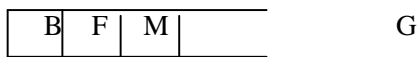
Step 3: POP the top node from the stack i.e. A and find the neighbours of A i.e. B and C.



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



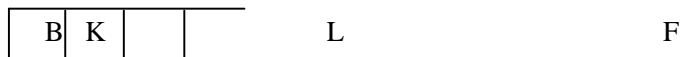
Step 5: Now G is the top node of the stack. Find its neighbour i.e. M



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



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



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



Also you can traverse the graph starting from the root A and then insert in the order C and B into the stack. Check your answer.

Advantages:

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

Disadvantages:

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

Brute Force or Blind Search

Brute force or blind search is a uniformed exploration of the search space and it does not explicitly take into account either planning efficiency or execution efficiency. Blind search is also called uniform search. It is the search which has no information about its domain. The only thing that a blind search can do is to differentiate between a non goal state and a goal state. These methods do not need domain knowledge but they are less efficient in result. Uniform strategies don't use any information about how close a node might be to a goal. They differ in the order that the nodes are expanded. The most important brute force techniques are breadth first search, depth first search, uniform search and bidirectional search. All brute force techniques must take (b^d) time and use $O(d)$ space. This technique is not as efficient as compared to other algorithms.

Difference between BFS and DFS

BFS

- ⇒ It uses the data structure queue.
- ⇒ BFS is complete because it finds the solution if one exists.
- ⇒ BFS takes more space i.e. equivalent to $O(b^d)$ where b is the maximum breadth exist in a search tree and d is the maximum depth exist in a search tree.
- ⇒ In case of several goals, it finds the best one.

DFS

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

Greedy Search

This algorithm uses an approach which is quite similar to the best first search algorithm. It is a simple best first search which reduces the estimated cost of reach the goal. Basically it takes the closest node that appears to be closest to the goal. This search starts with the initial matrix and makes very single possible changes then looks at the change it made to the score. This search then applies the change till the greatest improvement. The search continues until no further improvement can be made. The greedy search never

never makes a lateral move. It uses minimal estimated cost $h(n)$ to the goal state as a measure which decreases the search time but the algorithm is neither complete nor optimal. The main advantage of this search is that it is simple and finds a solution quickly. The disadvantages are that it is not optimal, susceptible to false start.

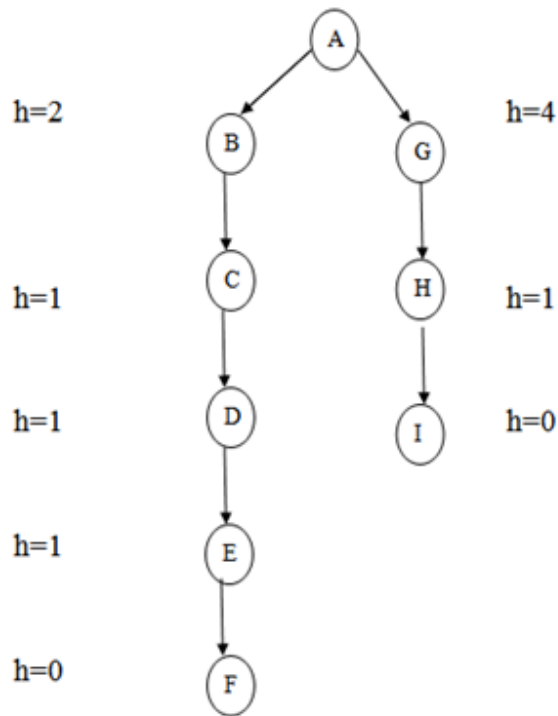


Figure Greedy Search

INFORMED SEARCH (HEURISTIC SEARCH)

Heuristic is a technique which makes our search algorithm more efficient. Some heuristics help to guide a search process without sacrificing any claim to completeness and some sacrificing it. Heuristic is a problem specific knowledge that decreases expected search efforts. It is a technique which sometimes works but not always. Heuristic search algorithm uses information about the problem to help directing the path through the search space. These searches use some functions that estimate the cost from the current state to the goal presuming that such function is efficient. A heuristic function is a function that maps from problem state descriptions to a measure of desirability usually represented as a number. The purpose of a heuristic function is to guide the search process in the most profitable directions by suggesting which path to follow first when more than one is available.

Generally, a heuristic incorporates domain knowledge to improve efficiency over blind search. In AI, a heuristic has a general meaning and also a more specialized technical meaning. Generally, a term heuristic is used for any advice that is effective but is not guaranteed to work in every case. For example, in the case of

travelling sales man (TSP) problem we are using a heuristic to calculate the nearest neighbour. Heuristic is a method that provides a better guess about the correct choice to make at any junction that would be achieved by random guessing. This technique is useful in solving tough problems which could not be solved in any other way. Solutions take an infinite time to compute.

Let us see some classifications of heuristic search.

Best First Search

Best first search is an instance of graph search algorithm in which a node is selected for expansion based on evaluation function $f(n)$. Traditionally, the node which is the lowest evaluation is selected for the explanation because the evaluation measures distance to the goal. Best first search can be implemented within general search frame work via a priority queue, a data structure that will maintain the fringe in ascending order of f values. This search algorithm serves as combination of depth first and breadth first search algorithm. Best first search algorithm is often referred greedy algorithm this is because they quickly attack the most desirable path as soon as its heuristic weight becomes the most desirable.

Concept:

Step 1: Traverse the root node

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

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

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

Algorithm:

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

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

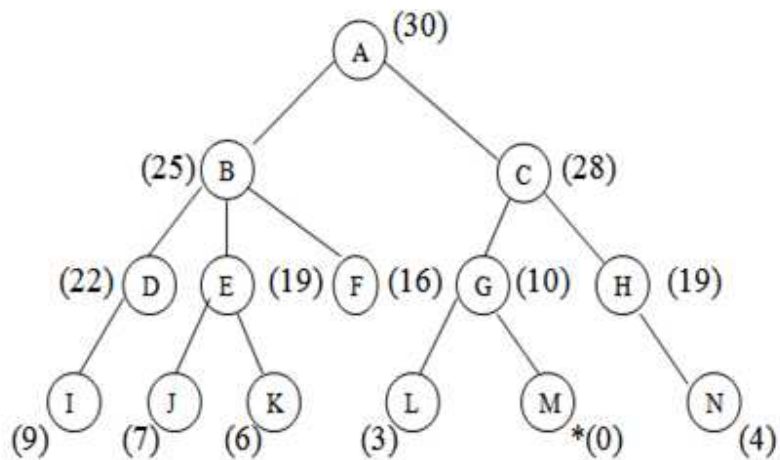
Step 3: If the first element of the queue is our goal node, then stop and return success.

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

Step 5: Go to step-3

Step 6: Exit.

Implementation:

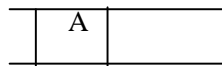


Let us solve an example for implementing above BFS algorithm.

Figure

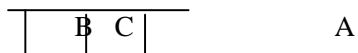
Step 1:

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



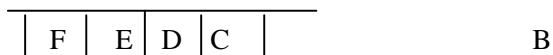
Step 2:

The neighbours of A are B and C. They will be inserted into the queue in ascending order.



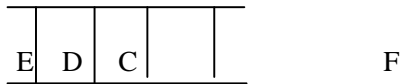
Step 3:

Now B is on the FRONT end of the queue. So calculate the neighbours of B that are maintaining a least distance from the roof.



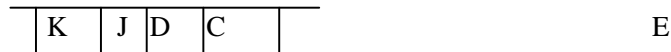
Step 4:

Now the node F is on the FRONT end of the queue. But as it has no further children, so remove it from the queue and proceed further.



Step 5:

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



Step 6:

Now K is on the FRONT end and as it has no further children, so remove it and proceed further



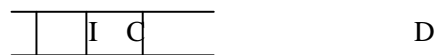
Step7:

Also, J has no corresponding children. So remove it and proceed further.



Step 8:

Now D is on the FRONT end and calculates the children of D and put it into the queue.



Step9:

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



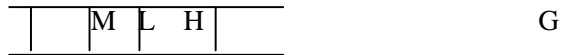
Step 10:

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



Step 11:

Now remove G from the queue and calculate its neighbour that is to insert in ascending order into the queue.

**Step12:**

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

**Advantage:**

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

Disadvantages:

Sometimes, it covers more distance than our consideration.

Branch and Bound Search

Branch and Bound is an algorithmic technique which finds the optimal solution by keeping the best solution found so far. If partial solution can't improve on the best it is abandoned, by this method the number of nodes which are explored can also be reduced. It also deals with the optimization problems over a search that can be presented as the leaves of the search tree. The usual technique for eliminating the sub trees from the search tree is called pruning. For Branch and Bound algorithm we will use stack data structure.

Concept:

Step 1: Traverse the root node.

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

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

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

Algorithm:

Step 1: PUSH the root node into the stack.

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

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

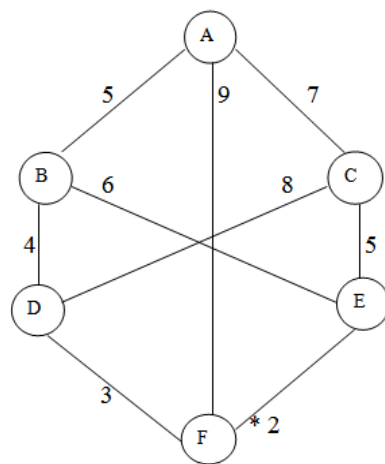
Step 4: Else POP the node from the stack. Process it and find all its successors. Find out the path containing all its successors as well as predecessors and then PUSH the successors which are belonging to the minimum or shortest path.

Step 5: Go to step 5.

Step 6: Exit.

Implementation:

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



Figure

Step 1:

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

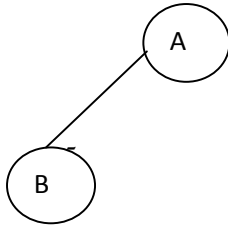
A	
---	--

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

F: $0+9 = 9$

C: $0+7 = 7$

Here B (5) is the least distance.



Step 2:

Now the stack will be



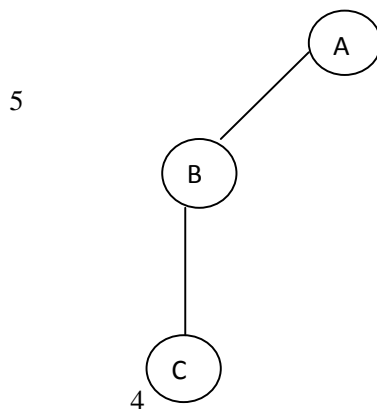
A

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

D: $0+5+4 = 9$

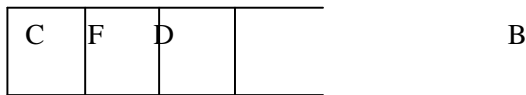
E: $0+5+6 = 11$

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



Step 3:

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



$$C: 0+5+4+8 = 17$$

$$F: 0+5+4+3 = 12$$

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

Step 4:



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

Advantages:

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

Disadvantages:

- ⇒ The load balancing aspects for Branch and Bound algorithm make it parallelization difficult.
- ⇒ The Branch and Bound algorithm is limited to small size network. In the problem of large networks, where the solution search space grows exponentially with the scale of the network, the approach becomes relatively prohibitive.

A* SEARCH

A* is a cornerstone name of many AI systems and has been used since it was developed in 1968 by Peter Hart; Nils Nilsson and Bertram Raphael. It is the combination of Dijkstra's algorithm and Best first search. It can be used to solve many kinds of problems. A* search finds the shortest path through a search space to goal state using heuristic function. This technique finds minimal cost solutions and is directed to a goal state called A* search. In A*, the * is written for optimality purpose. The A* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. This algorithm is complete if the branching factor is finite and every action has fixed cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. It can be defined by following formula.

$$f(n) = g(n) + h(n)$$

Where

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

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

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

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

OPEN:

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

CLOSE:

An array which contains the nodes that have been examined.

Algorithm:

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

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

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

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

Step 5: Go to Step-2.

Step 6: Exit.

Implementation:

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

Advantages:

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

Disadvantages:

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

AO* Search: (And-Or) Graph

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

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

OPEN:

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

CLOSE:

It contains the nodes that have already been processed.

$h(n)$:The distance from current node to goal node.

Algorithm:

Step 1: Place the starting node into OPEN.

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

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

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

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

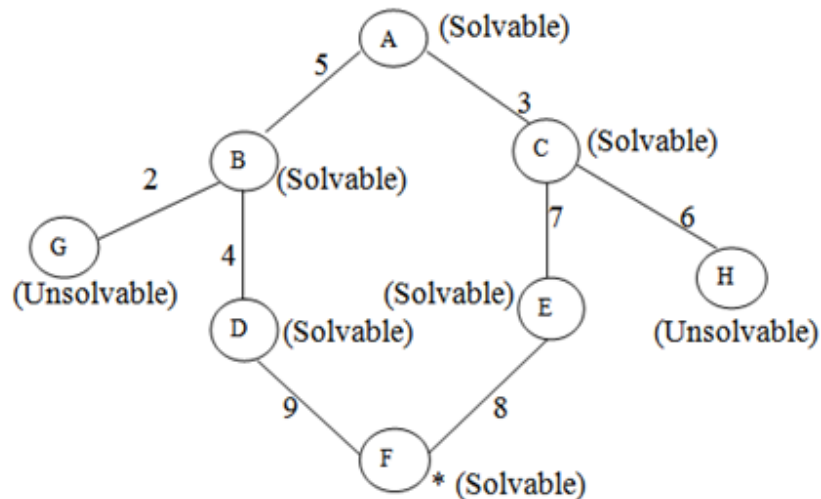
Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit.

Implementation:

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



Figure

Step 1:

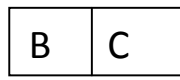
In the above graph, the solvable nodes are A, B, C, D, E, F and the unsolvable nodes are G, H. Take A as the starting node. So place A into OPEN.

i.e. OPEN = A CLOSE = (NULL) ϕ A

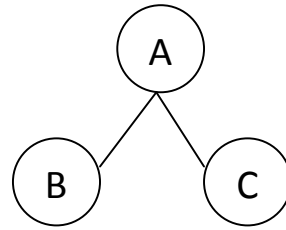
Step 2:

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

i.e. OPEN =



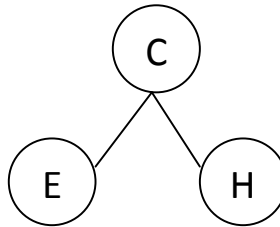
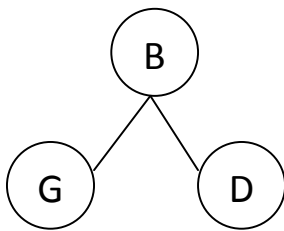
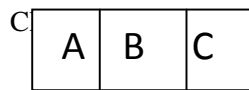
CLOSE =



Step 3:

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

So OPEN =



(O)

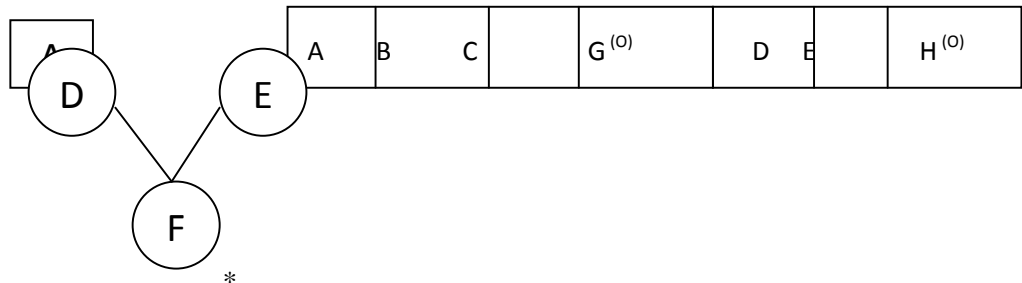
'O' indicated that the nodes G and H are unsolvable.

Step 4:

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

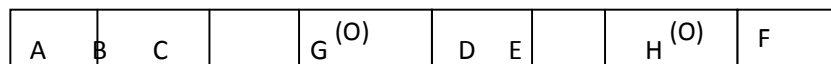
i.e. OPEN =

CLOSE =



Step 5:

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

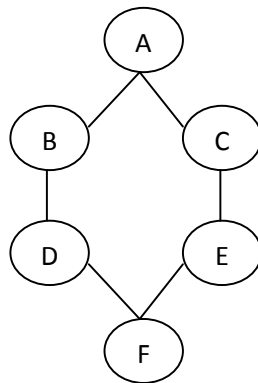


i.e. CLOSE =

Step 6:

Success and Exit

AO* Graph:



Figure

Advantages:

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

Disadvantages:

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

Hill Climbing

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

The name hill climbing is derived from simulating the situation of a person climbing the hill. The person will try to move forward in the direction of at the top of the hill. His movement stops when it reaches at

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

Algorithm:

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

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

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

Step 4:

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

Step 5: Exit.

Advantages:

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

Disadvantages:

The question that remains on hill climbing search is whether this hill is the highest hill possible. Unfortunately without further extensive exploration, this question cannot be answered. This technique works but as it uses local information that's why it can be fooled. The algorithm doesn't maintain a search tree, so the current node data structure need only record the state and its objective function value. It assumes that local improvement will lead to global improvement.

There are some reasons by which hill climbing often gets stuck which are stated below.

Local Maxima:

A local maxima is a state that is better than each of its neighbouring states, but not better than some other states further away. Generally this state is lower than the global maximum. At this point, one cannot decide easily to move in which direction! This difficulties can be extracted by the process of backtracking i.e. backtrack to any of one earlier node position and try to go on a different event direction. To implement this strategy, maintaining in a list of path almost taken and go back to one of them. If the path was taken that leads to a dead end, then go back to one of them.

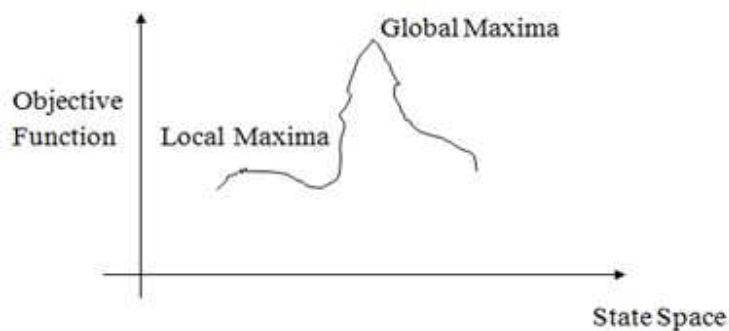


Figure Local Maxima

Ridges:

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

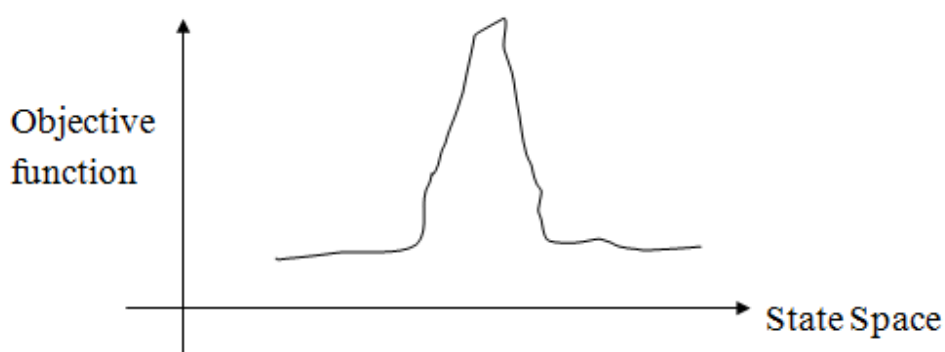


Figure Ridges

Plateau:

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

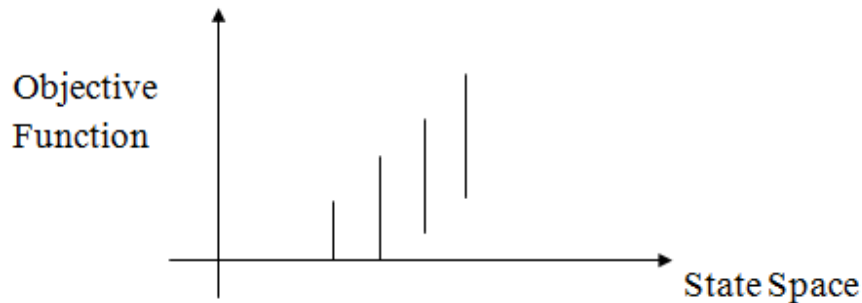


Figure Plateau

KNOWLEDGE

Knowledge is the collection of facts, inference rules etc. which can be used for a particular purpose. Knowledge requires the use of data and information. It combines relationships, correlations, dependencies with data and information.

The basic components of knowledge are:

- 1) A set of collected data
- 2) A form of belief or hypothesis
- 3) A kind of information.

Knowledge is different from data. Data is the collection of raw materials where as knowledge is the collection of some well specified inference rules and facts. Knowledge is also different from belief and hypothesis. Belief is any meaningful and coherent expression that can be represented. Belief may be true or false. A hypothesis is a justified belief that is not known to be true. A hypothesis is a belief which is backed up with some supporting evidence but it may still be false. So knowledge can be defined as true justified knowledge.

KNOWLEDGE BASED SYSTEMS

Knowledge based systems get their power from the expert knowledge that has been coded into facts, rules, heuristics and procedures. The knowledge is stored in a knowledge base separate from the control and inferencing components. Knowledge is important and essential for knowledge based intelligent behaviour.

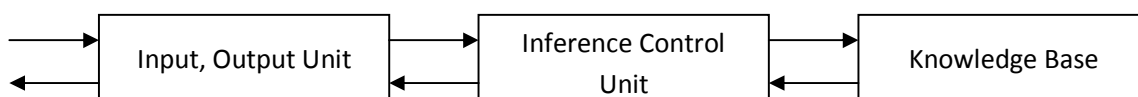


Figure A typical Knowledge based system

Any choice of representation will depend on the type of problem to be solved and the inference methods available. Knowledge may be vague, contradictory or incomplete. Thus, knowledge is information about objects, concepts and relationships that are assumed to exist in a particular area of interest.

TYPE OF KNOWLEDGE

The categorisation of knowledge is very much large and interesting. They can be of following types:

Declarative knowledge

It is the passive knowledge expressed as statements of facts about the world. It gives the simple facts and ideas about any phenomenon. It means just the representation of facts or assertions. This tells the total description about the situation. For example, the facts about an organization may be its buildings, location, no. of departments, no. of employees etc. The facts may be of two types i.e. static and dynamic. The static facts do not change with time where as the dynamic facts change with time. For example, the name and location of an organization is permanent. But some additional departments may be added.

Procedural knowledge

Procedural knowledge is the compiled knowledge related to the performance of some task. For example the steps used to solve an algebraic equation can be expressed as procedural knowledge. It also eradicates the limitations of declarative knowledge i.e. declarative knowledge tells about the organization but it cannot tell how the employees are working in that organization and how the products are developed. But procedural knowledge describes everything about the organization by using production rules and dynamic attributes.

For example, If: All the employees are very hardworking

They are very punctual

They have productive ideas.

Then: Large no. of products can be produced within a very limited time period.

The advantages of using procedural knowledge are as follows:

- 1) Domain specific knowledge can be easily represented.
- 2) Extended logical inferences, such as default reasoning facilitated.

3) Side effects of actions may be modeled.

Some disadvantages of procedural knowledge are

- 1) Completeness: In procedural knowledge not all cases may be represented.
- 2) Consistency: Not all deductions may be correct.
- 3) Modularity: Changes in knowledge base might have far-reaching effects.

Inheritable knowledge

There are many situations in the world, where the object of an event inherits some properties of that particular event or any other event.

For example, consider a college. A college has certain features like classrooms, teachers, play ground, furniture, students etc. Besides these, there will be some general concepts regarding the functioning of the college, like it will have time table for each class, a fee deposit plan, examination pattern, course module etc. It can have many more deep concepts like placement of students etc. Now, if we say “A is a College”, then A will automatically inherits all the features of the college. It may be possible that X has some additional features. The inheritable knowledge is diagrammatically represented below. Here, the relationship ‘has’ indicates the silent features or attributes and ‘is a’ represents the variable or instance of that type. A inherits all the properties of college and has one additional feature of having male students. In this type of knowledge, data must be organized into a hierarchy of classes. The arrows represent the point from object to its value in the diagram. Boxed nodes represent the objects and values of attributes of objects.

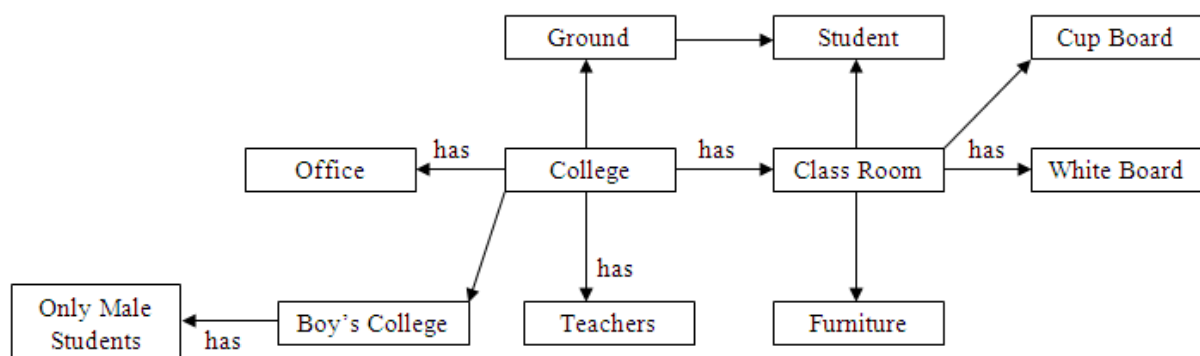


Figure College Attribute representation

Relational Knowledge

Relational knowledge is made up of objects consisting of attributes and corresponding associated values. In this type of knowledge, the facts are represented as set of relations in a tabular form. The table stores or captures all the hidden attributes of objects.

For example the knowledge about doctors may be as mentioned in figure .

Department	Qualification	Height	Age
Eye	P.HD	5.0	35
Kidney	P.HD	5.10	32
Surgery	P.HD	6.3	28
Medicine	P.HD	6.1	44

Figure Knowledge about Doctor

This form of representation is the simplest and can be used in database systems. But this representation cannot store any semantic, information. For example, from this information we cannot answer the questions like “What is the name of the doctor”? or “How many doctors are in eye department”?

Inferential Knowledge

The knowledge, which can use inference mechanism to use this knowledge is called inferential knowledge. The inheritance property is a very powerful form of inferential knowledge. The inference procedures implement the standard logic rules of inference. There are two types of inference procedures like forward inference and backward inference. Forward inference moves from start state to goal state whereas backward inference moves from goal state to start state. In this type of knowledge several symbols are generally used like \forall (universal quantifier), \exists (existential quantifier), \rightarrow (arrow indicator) etc.

For example: All cats have tails

$$\forall X: \text{cat}(x) \rightarrow \text{has tail}(x)$$

Advantages:

- 1) A set of strict rules are defined which can be used to derive more facts.
- 2) Truths of new statements can be verified.
- 3) It gives guarantee about the correctness.
- 4) Many inference procedures available to implement standard rules of logic.

Heuristic Knowledge

This type of knowledge is fully experimental. This knowledge requires some judgments about any performance. One can guess a good thing and also one can think bad thing. But good performances are generally taken in heuristic knowledge. For example, suppose it is asked that “Ram will score how much percentage in his final semester?” Then the answer might be 80%, 70%, 30% or 95%. The individual answers of this question based on the heuristic knowledge. The answer would be based on various factors such as past performance, his talent etc. If his previous semester percentage was 78%, then if one will say he will secure 10% in this semester then obviously he has not any knowledge about Ram.

Tacit Knowledge

This kind of knowledge is acquired by experience. Tacit knowledge is subconsciously understood and applied, difficult to articulate and formalize. This type of knowledge is developed from direct experience and action. This knowledge is usually shared through highly interactive conversation, story telling and experience. It also includes cognitive skills such as intuition as well as technical skills such as craft and know-how. Tacit knowledge cannot be transmitted before it is converted into words, models or numbers that can be understood. Tacit knowledge can be defined in two dimensions, such as technical dimension and cognitive dimension. In technical dimension highly subjective and personal insights, intuitions and inspirations derived from long experience. The dimensions such as beliefs, ideals, principles, values and emotions fall in the category of cognitive dimension.

Explicit Knowledge

This knowledge is formalized, coded in several natural languages (English, Italian and Spanish) or artificial languages (UML, Mathematics etc). This knowledge can be easily transmitted. It includes theoretical approaches, problem solving, manuals and database. As explicit knowledge, it was the first to be or, at least, to be archived. Tacit and explicit knowledge are not totally separate, but mutually complementary entities. Without any experience, we cannot truly understand. Explicit knowledge is playing an increasingly large role in organization and it is considered by some to be the most important factor of production in the knowledge economy. Imagine an organization without procedure manuals product literature or computer software. Also with explicit knowledge, some tacit knowledge is required to run the business in an organization. Without explicit knowledge, the organization is simply has a zero performance.

Research Knowledge

There are many standards for the generation and critical appraisal of research knowledge, but judging the quality of knowledge in this source is not without difficulty. There are disputes about the nature and content of standards in areas such as qualitative research, and the implementation of standards is

sometimes weak so that conformity with them is not necessarily a guarantee of quality. This type of knowledge is very useful for researchers to improve the research quality.

KNOWLEDGE ACQUISITION

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

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

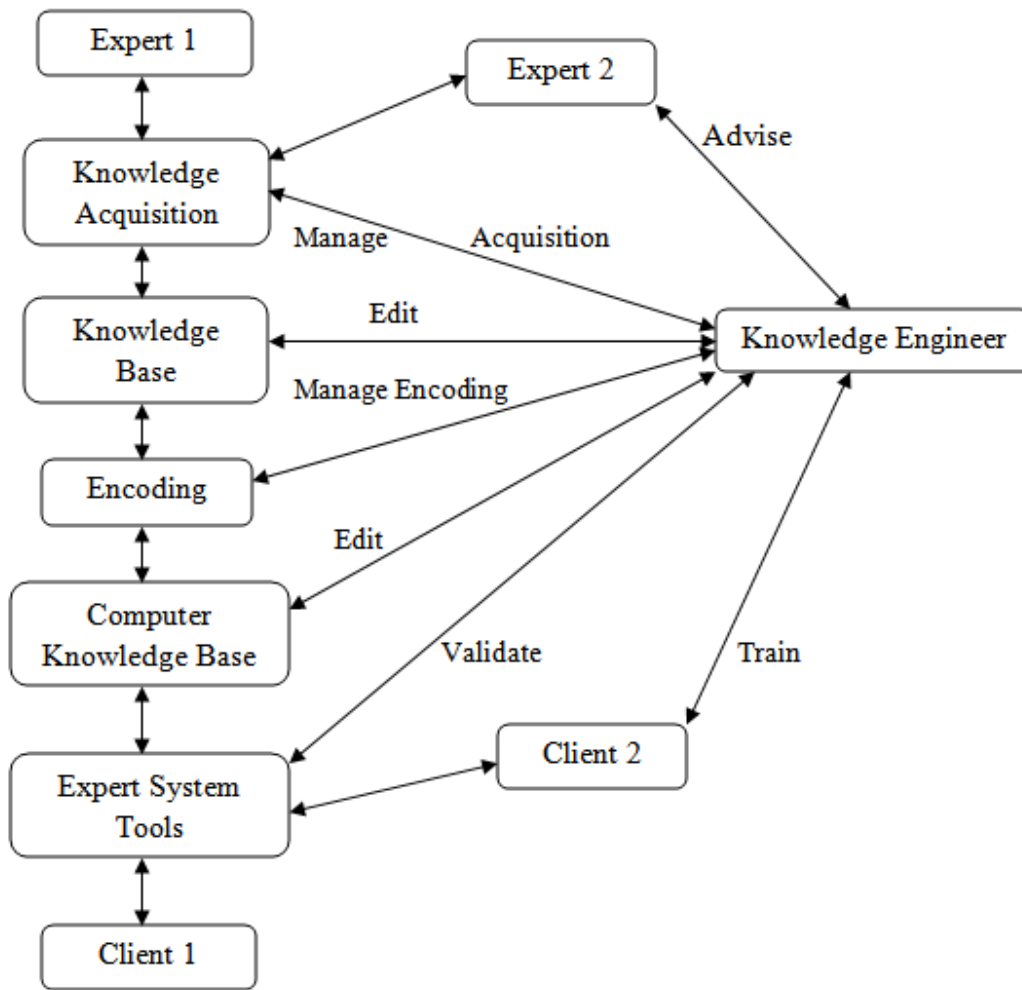


Figure Knowledge Engineer's Roles in Interactive Knowledge Acquisition

Knowledge Acquisition Techniques

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

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

In diagram based techniques the generation and use of concept maps, event diagrams and process maps. This technique captures the features like "why, when, who, how and where". The matrix based techniques involve the construction of grids indicating such things as problems encountered against possible

solutions. Hierarchical techniques are used to build hierarchical structures like trees. Protocol analysis technique is used to identify the type of knowledge like goals, decisions, relationships etc. The protocol generation techniques include various types of interviews like structured, semi-structured and unstructured.

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

KNOWLEDGE REPRESENTATION

Knowledge representation is probably, the most important ingredient for developing an AI. A representation is a layer between information accessible from outside world and high level thinking processes. Without knowledge representation it is impossible to identify what thinking processes are, mainly because representation itself is a substratum for a thought.

The subject of knowledge representation has been messaged for a couple of decades already. For many applications, specific domain knowledge is required. Instead of coding such knowledge into a system in a way that it can never be changed (hidden in the overall implementation), more flexible ways of representing knowledge and reasoning about it have been developed in the last 10 years.

The need of knowledge representation was felt as early as the idea to develop intelligent systems. With the hope that readers are well conversant with the fact by now, that intelligent requires possession of knowledge and that knowledge is acquired by us by various means and stored in the memory using some representation techniques. Putting in another way, knowledge representation is one of the many critical aspects, which are required for making a computer behave intelligently. Knowledge representation refers to the data structures techniques and organizing notations that are used in AI. These include semantic networks, frames, logic, production rules and conceptual graphs.

Properties for knowledge Representation

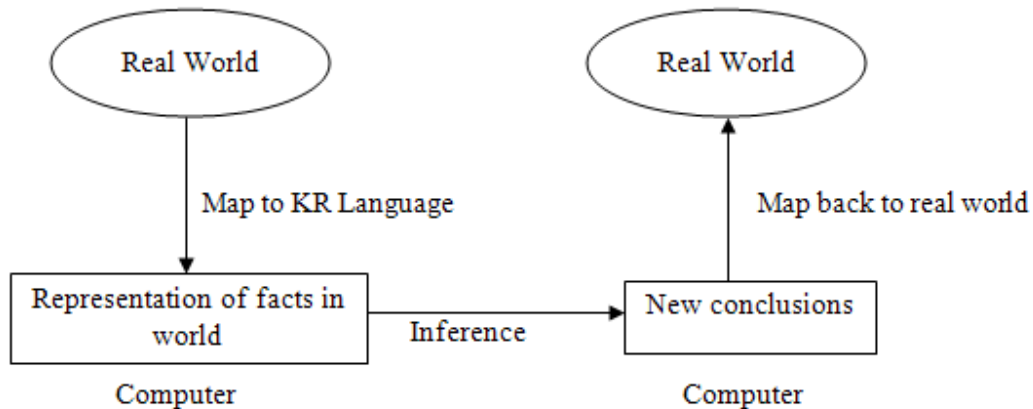
The following properties should be possessed by a knowledge representation system.

- a. **Representational Adequacy:** It is the ability to represent the required knowledge.
- b. **Inferential Adequacy:** It is the ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original.
- c. **Inferential Efficiency:** The ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides.
- d. **Acquisitional Efficiency:** The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

Syntax and semantics for Knowledge Representation

Knowledge representation languages should have precise syntax and semantics. You must know exactly what an expression means in terms of objects in the real world. Suppose we have decided that “red 1” refers to a dark red colour, “car1” is my car, car2 is another. Syntax of language will tell you which of the following is legal: red1 (car1), red1 car1, car1 (red1), red1 (car1 & car2)?

Semantics of language tell you exactly what an expression means: for example, Pred (Arg) means that the property referred to by Pred applies to the object referred to by Arg. E.g., properly “dark red” applies to my car.



Types of Knowledge Representation

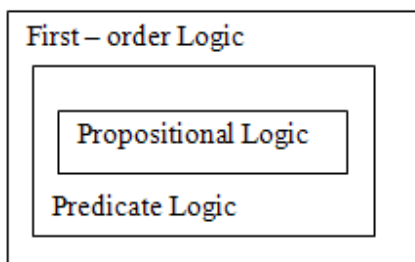
Knowledge can be represented in different ways. The structuring of knowledge and how designers might view it, as well as the type of structures used internally are considered. Different knowledge representation techniques are

- a. Logic
- b. Semantic Network
- c. Frame

- d. Conceptual Graphs
- e. Conceptual Dependency
- f. Script

Logic

A logic is a formal language, with precisely defined syntax and semantics, which supports sound inference. Different logics exist, which allow you to represent different kinds of things, and which allow more or less efficient inference. The logic may be different types like propositional logic, predicate logic, temporal logic, description logic etc. But representing something in logic may not be very natural and inferences may not be efficient.



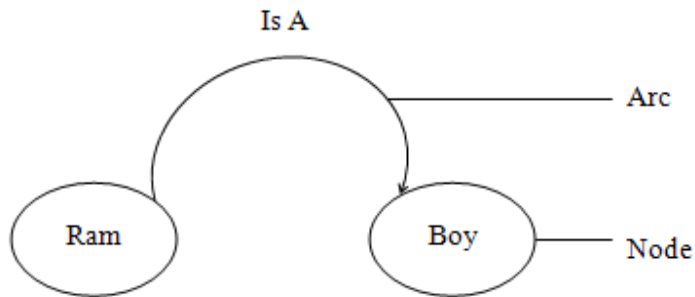
Figure

Semantic Network

A semantic network is a graphical knowledge representation technique. This knowledge representation system is primarily on network structure. The semantic networks were basically developed to model human memory. A semantic net consists of nodes connected by arcs. The arcs are defined in a variety of ways, depending upon the kind of knowledge being represented.

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. The semantic network consists of different nodes and arcs. Each node should contain the information about objects and each arc should contain the relationship between objects. Semantic nets are used to find relationships among objects by spreading activation about from each of two nodes and seeing where the activation met this process is called intersection search.

For example: Ram is a boy.



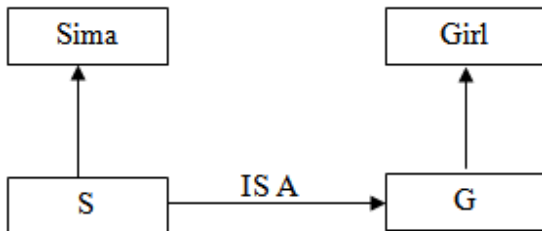
Figure

Semantic network by using Instances

The semantic network based knowledge representation mechanism is useful where an object or concept is associated with many attributes and where relationships between objects are important. Semantic nets have also been used in natural language research to represent complex sentences expressed in English. The semantic representation is useful because it provides a standard way of analyzing the meaning of sentence. It is a natural way to represent relationships that would appear as ground instances of binary predicates in predicate logic. In this case we can create one instance of each object. In instance based semantic net representations some keywords are used like: IS A, INSTANCE, AGENT, HAS-PARTS etc.

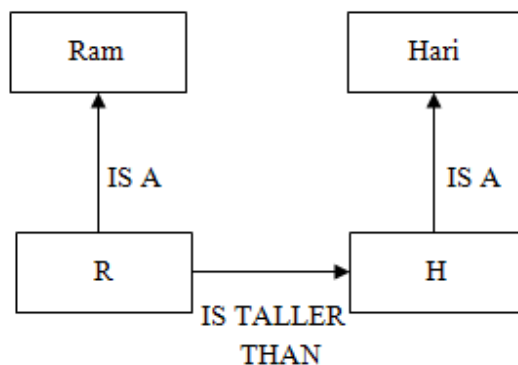
Consider the following examples:

1. Suppose we have to represent the sentence "Sima is a girl".

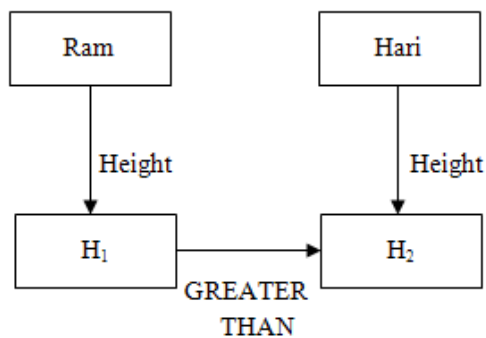


Figure

2. Ram is taller than Hari



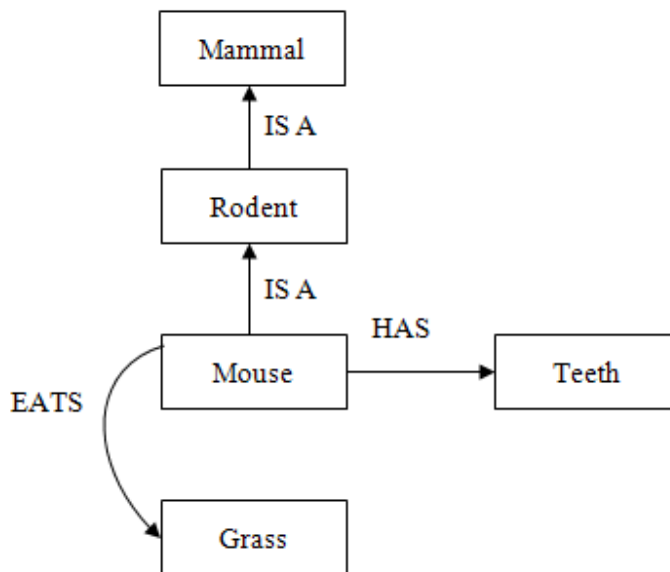
It can also be represented as



(b)

3. “Mouse is a Rodent and Rodent is a mammal. Mouse has teeth and etas grass”. Check whether the sentence mammal has teeth is valid or not.]

(c)



Partitioned Semantic Network

Some complex sentences are there which cannot be represented by simple semantic nets and for this we have to follow the technique partitioned semantic networks. Partitioned semantic net allow for

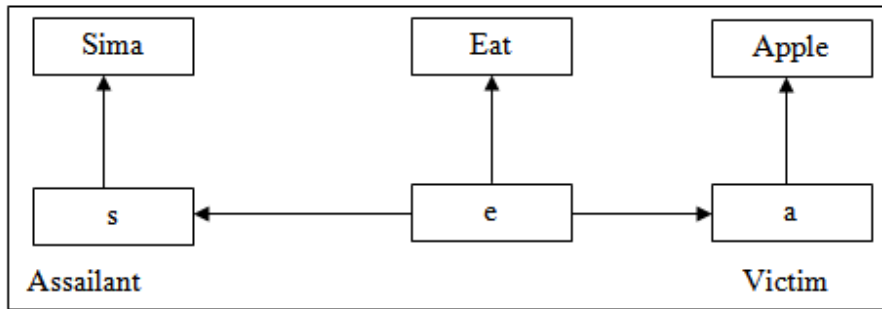
1. Propositions to be made without commitment to truth.
2. Expressions to be quantified.

In partitioned semantic network, the network is broken into spaces which consist of groups of nodes and arcs and regard each space as a node.

Let us consider few examples.

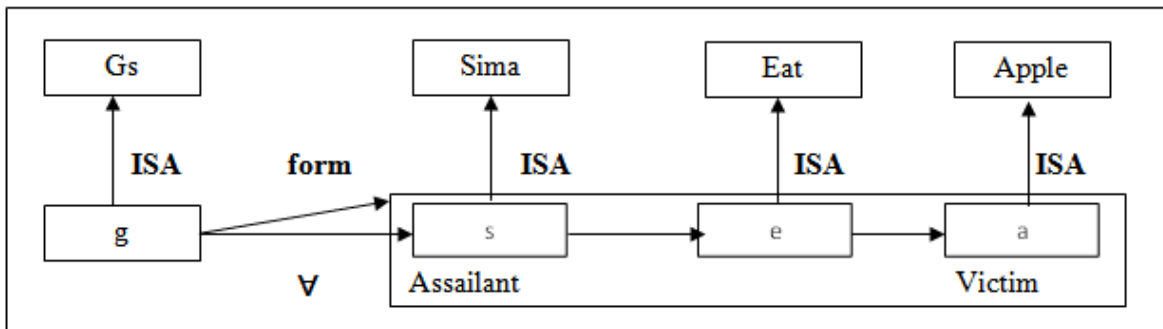
Draw the partitioned semantic network structure for the followings:

- a) Sima is eating an apple.



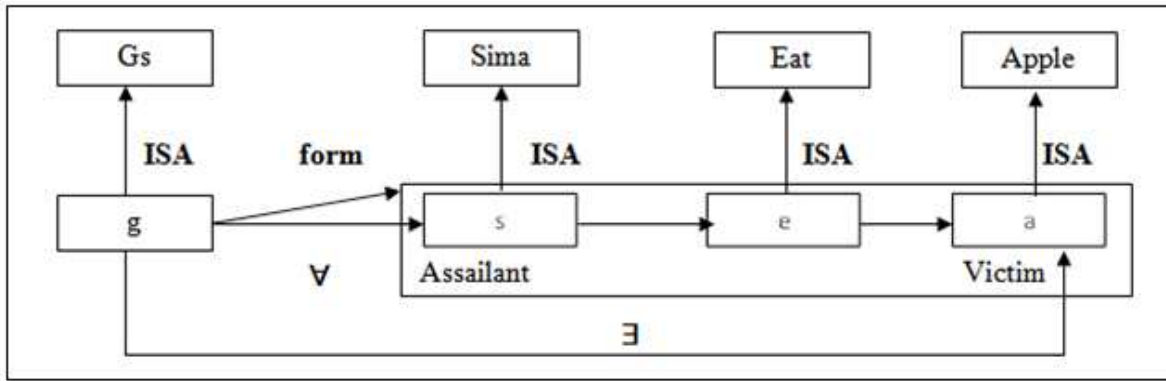
Figure

- b) All Sima are eating an apple.



Figure

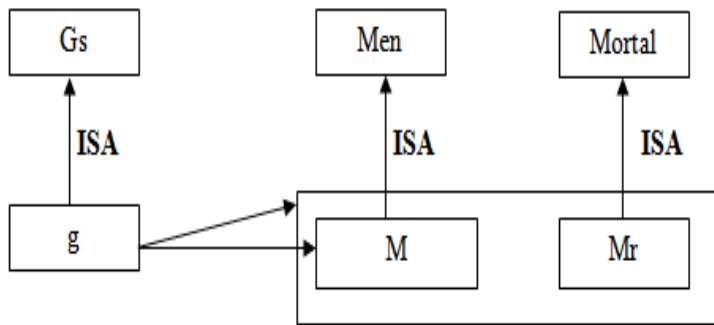
- c) All Sima are eating some apple.



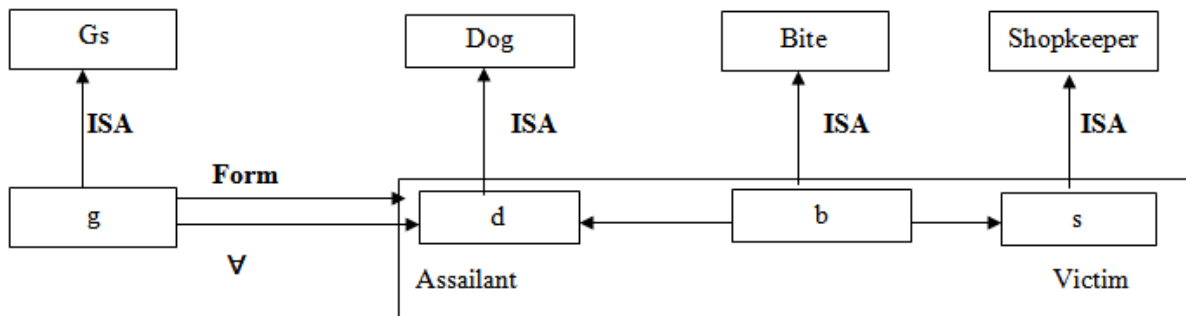
Figure

d) All men are mortal

Figure

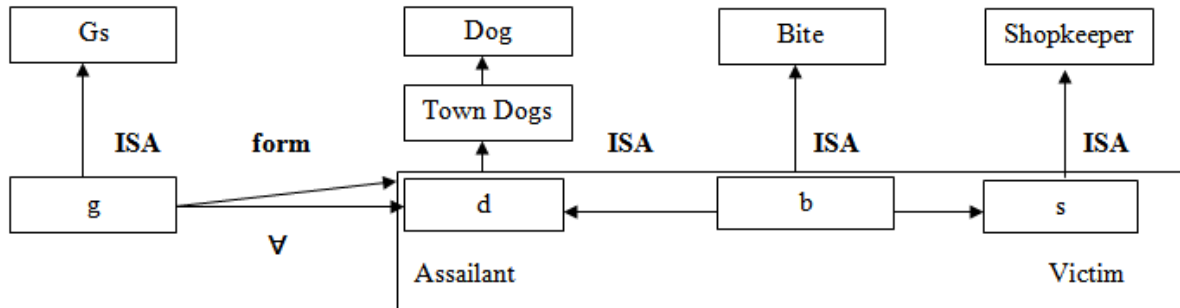


e) Every dog has bitten a shopkeeper



Figure

f) Every dog in town has bitten a shopkeeper.



Figure

NOTE: On the above semantic network structures, the instance “IS A” is used. Also two terms like assailant and victim are used. Assailant means “by which the work is done” and that of victim refers to “on which the work is applied”. Another term namely GS, which refers to General Statement. For GS, make a node g which is an instance of Gs. Every element will have at least two attributes. Firstly, a form that states which a relation is being asserted. Secondly, one or more for all (\forall) or there exists (\exists) connections which represent universally quantifiable variables.

FRAME

A frame is a collection of attributes and associated values that describe some entity in the world. Frames are general record like structures which consist of a collection of slots and slot values. The slots may be of any size and type. Slots typically have names and values or subfields called facets. Facets may also have names and any number of values. A frame may have any number of slots, a slot may have any number of facets, each with any number of values. A slot contains information such as attribute value pairs, default values, condition for filling a slot, pointers to other related frames and procedures that are activated when needed for different purposes. Sometimes a frame describes an entity in some absolute sense, sometimes it represents the entity from a particular point of view. A single frame taken alone is rarely useful. We build frame systems out of collection 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. Each frame should start with an open parenthesis and closed with a closed parenthesis.

Syntax of a frame


```

(<frame name>
  (<slot 1> (<facet1> <value 1>.....<value n1>)
            (<facet2> <value1>.....<value n2>)
            .
            .
            .
            .
            .
            (<facet n> <value1>..... <value nn>))
  (<slot 2> (<facet1> <value 1>.....<value n1>)
            (<facet2><value2>.....<value n2>)
            .
            .
            .
  ))

```

Let us consider the below examples.

1) Create a frame of the person Ram who is a doctor. He is of 40. His wife name is Sita. They have two children Babu and Gita. They live in 100 kps street in the city of Delhi in India. The zip code is 756005.

```

(Ram
  (PROFESSION (VALUE Doctor))
  (AGE (VALUE 40))
  (WIFE (VALUE Sita))
  (CHILDREN (VALUE Bubu, Gita))
  (ADDRESS
    (STREET (VALUE 100 kps))
    (CITY(VALUE Delhi))
    (COUNTRY(VALUE India))
    (ZIP (VALUE 756005))))

```

2) Create a frame of the person Anand who is a chemistry professor in RD Women's College. His wife name is Sangita having two children Rupa and Shipa.

(Anand

(PROFESSION (VALUE Chemistry Professor))

(ADDRESS (VALUE RD Women's College))

(WIFE (VALUE Sangita))

(CHILDREN(VALUE RupaShipa)))

3) Create a frame of the person Akash who has a white maruti car of LX-400 Model. It has 5 doors. Its weight is 225kg, capacity is 8, and mileage is 15 km /lit.

(Akash

(CAR (VALUE Maruti))

(COLOUR (VALUE White))

(MODEL (VALUE LX-400))

(DOOR (VALUE 5))

(WEIGHT (VALUE 225kg))

(CAPACITY (VALUE 8))

(MILAGE (VALUE 15km/lit)))

The frames can be attached with another frame and can create a network of frames. The main task of action frame is to provide the facility for procedural attachment and help in reasoning process. Reasoning using frames is done by instantiation. Instantiation process begins, when the given situation is matched with frames that are already in existence. The reasoning process tries to match the current problem state with the frame slot and assigns them values. The values assigned to the slots depict a particular situation and by this, the reasoning process moves towards a goal. The reasoning process can be defined as filling slot values in frames.

Conceptual Graphs

It is a knowledge representation technique which consists of basic concepts and the relationship between them. As the name indicates, it tries to capture the concepts about the events and represents them in the form of a graph. A concept may be individual or generic. An individual concept has a type field followed by a reference field. For example person : Ram. Here person indicates type and Ram indicates reference.

An individual concept should be represented within a rectangle in graphical representation and within a square bracket in linear representation. The generic concept should be represented within an oval in graphical representation and within a parenthesis in linear representation. Conceptual graph is a basic building block for associative network. Concepts like AGENT, OBJECT, INSTRUMENT, PART are obtained from a collection of standard concepts. New concepts and relations can be defined from these basic ones. These are also basic building block for associative network. A linear conceptual graph is an elementary form of this structure. A single conceptual graph is roughly equivalent to a graphical diagram of a natural language sentence where the words are depicted as concepts and relationships.

Consider an example

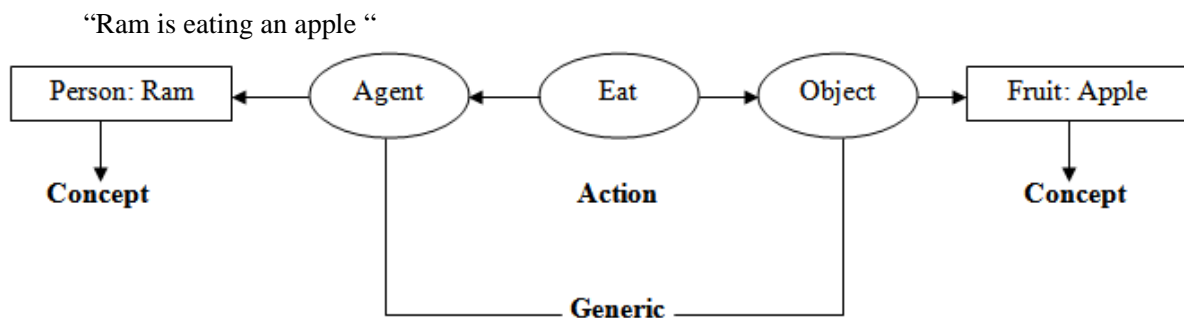
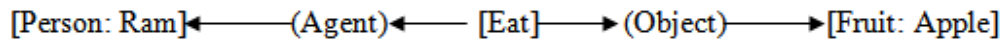


Figure Graphical Representation



Conceptual Dependency

It is another knowledge representation technique in which we can represent any kind of knowledge. It is based on the use of a limited number of primitive concepts and rules of formation to represent any natural language statement. Conceptual dependency theory is based on the use of knowledge representation methodology was primarily developed to understand and represent natural language structures. The conceptual dependency structures were originally developed by Roger C Schank in 1977.

If a computer program is to be developed that can understand wide phenomenon represented by natural languages, the knowledge representation should be powerful enough to represent these concepts. The conceptual dependency representation captures maximum concepts to provide canonical form of meaning of sentences. Generally there are four primitives from which the conceptual dependency structure can be described. They are

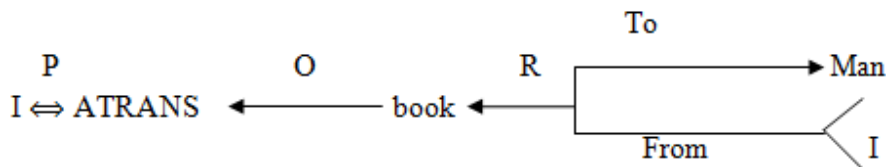
- a. ACTS : Actions
- b. PPs : Objects (Picture Producers)
- c. AAs : Modifiers of Actions (Action Aiders)

d. PAs : Modifiers of PPs (Picture Aiders)

e. TS : Time of action

Conceptual dependency provides both a structure and a specific set of primitives at a particular level of granularity, out of which representation of particular pieces of information can be constructed.

For example



Where ←: Direction of dependency

Double arrow indicates two way link between actor and action.

P: Past Tense

ATRANS: One of the primitive acts used by the theory

O: The objective case relation

R: Recipient case Relation

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

- 1) **ATRANS:** Transfer of an abstract relationship (give, accept, take)
- 2) **PTRANS:** Transfer the physical location of an object (Go, Come, Run, Walk)
- 3) **MTRANS:** Transfer the mental information (Tell)
- 4) **PROPEL:** Application of physical force to an object (push, pull, throw)
- 5) **MOVE:** Movement of a body part by its owner (kick).
- 6) **GRASP:** Grasping of an object by an action (clutch)
- 7) **INGEST:** Ingestion of an object by an animal (eat)
- 8) **EXPEL:** Expel from an animal body (cry)
- 9) **MBUILD:** Building new information out of old (decide)
- 10) **SPEAK:** Production of sounds (say)
- 11) **ATTEND:** Focusing of a sense organ towards a stimulus (Listen)

The main goal of CD representation is to capture the implicit concept of a sentence and make it explicit.

In normal representation of the concepts, besides actor and object, other concepts of time, location, source and destination are also mentioned. Following conceptual tenses are used in CD representation.

- 1) O : Object case relationship
- 2) R : Recipient case relationship
- 3) P : Past
- 4) F : Future
- 5) Nil : Present
- 6) T : Transition
- 7) Ts : Start Transition
- 8) Tf : Finisher Transition
- 9) K : Continuing
- 10) ? : Interrogative
- 11) / : Negative
- 12) C : Conditional

Also there are several rules in conceptual dependency

Rule 1: PP $\langle \overline{\text{ACT}} \rangle$

It describes the relationship between an actor and an event, he/she causes.

E.g. Ram ran

Ram $\langle \overline{\text{P}} \rangle$ PTRANS

Where P: Past Tense

Rule 2: PP $\langle \overline{\text{PA}} \rangle$

It describes the relationship between a PP and PA where the PA indicates one characteristics of PP. E.g.

Ram is tall

Ram $\langle \overline{\text{Nil}} \rangle$ Tall or Ram $\langle \overline{\text{Nil}} \rangle$ Height (> Average)

Rule 3: PP $\langle \overline{\text{PP}} \rangle$

It describes the relationship between two PPs where one PP is defined by other.

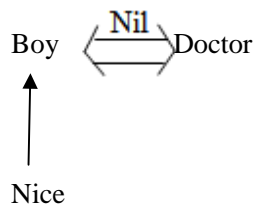
E.g. Ram is a doctor

Ram $\langle \overline{\text{Nil}} \rangle$ Doctor

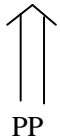
Rule 4: PP or PA
 \uparrow \downarrow
 PA PP

It describes the relationship between the PP and PA, where PA indicates one attributes of PP.

E.g. A nice boy is a doctor



Rule 5: PP



It describes the relationship between 3 PP's where one PP is the owner of another PP.

E.g. Ram's Cat

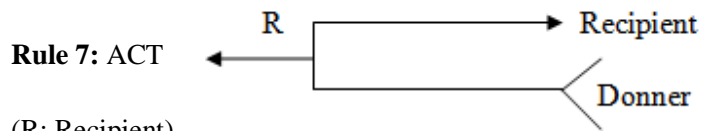
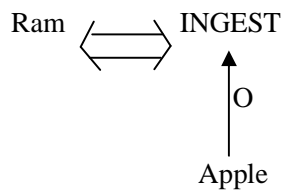


Ram

Rule 6: Act $\xleftarrow{\text{O}}$ PP Where O: Object

It describes the relationship between the PP and ACT. Where PP indicates the object of that action. E.g.

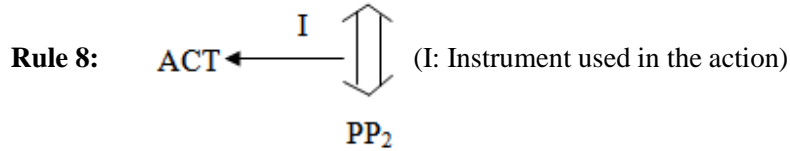
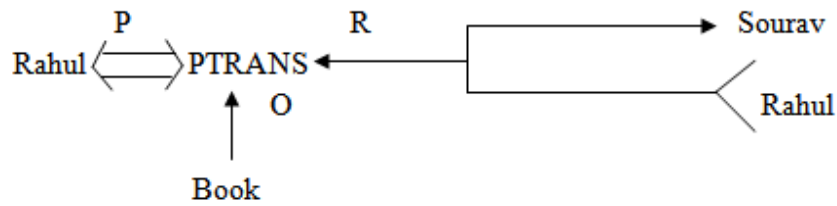
Ram is eating an apple.



(R: Recipient)

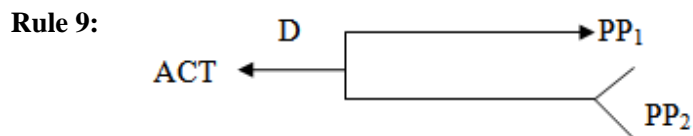
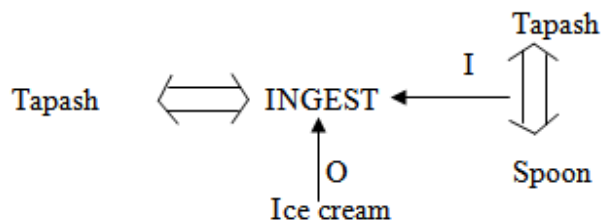
Here one PP describes the recipient and another PP describes the donner

E.g. Rahul gave a book to sourav.



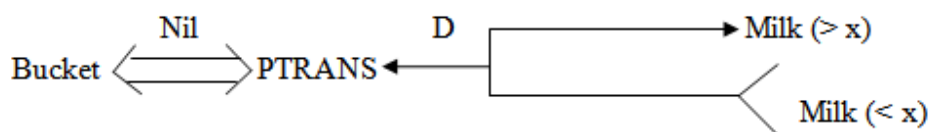
Here PP₁ indicates the agent and PP₂ indicates the object that is used in the action.

E.g. Tapash ate the ice cream with the spoon.

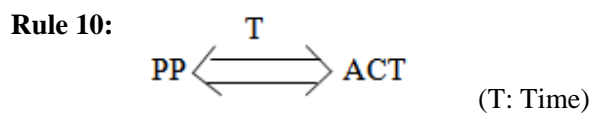


Here D indicates destination, PP₁ indicates destination and PP₂ indicates the source.

E.g. the bucket is filled with milk.

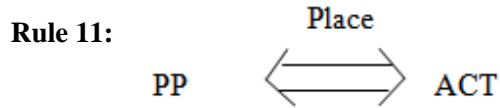
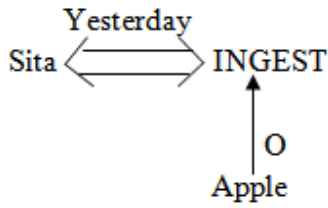


x indicates the average milk and the source i.e. bucket is dry which is hidden.



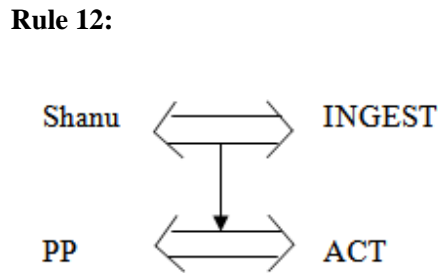
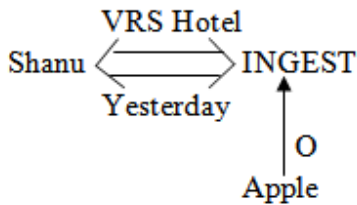
It describes the relationship between a conceptualization and the time at which the event is described occurs.

E.g. Sita ate the apple yesterday.



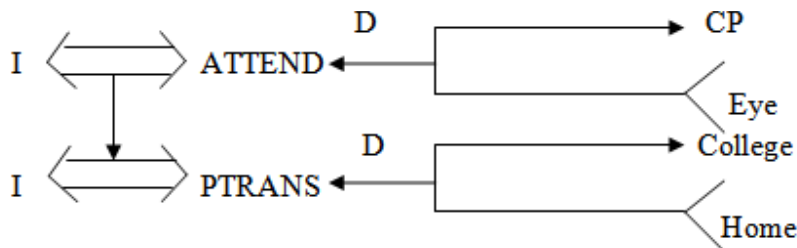
It describes the relationship between a conceptualization and the place at which it is occurred.

E.g. Shanu ate the apple at VRS hotel yesterday



It describes the relationship between one conceptualization with another.

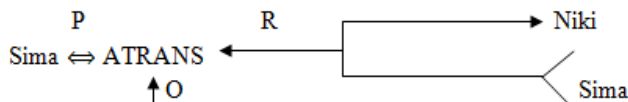
E.g. while I was going to college, I saw a snake



(Where CP: Conscious Processor i.e. the combination of all sense organs like eye, ear, nose etc.)

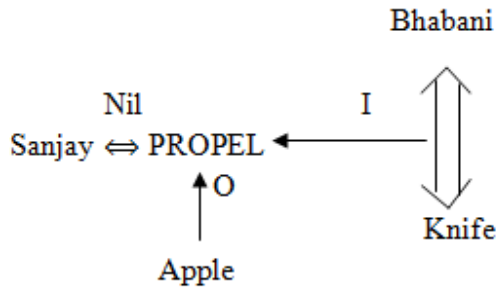
By using the above rules we can represent any sentence. Let us visualize few examples on conceptual dependency.

1) Sima gave a book to Niki

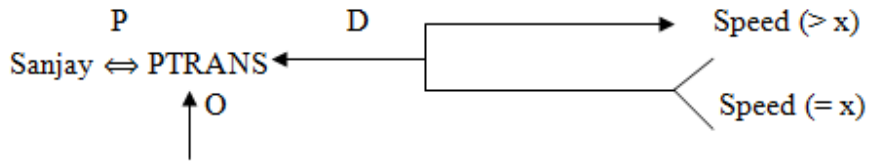


Where O: Object, P: Past Tense, R: Recipient, Sima: PP, Book: PP, Niki: PP, ATRANS: give

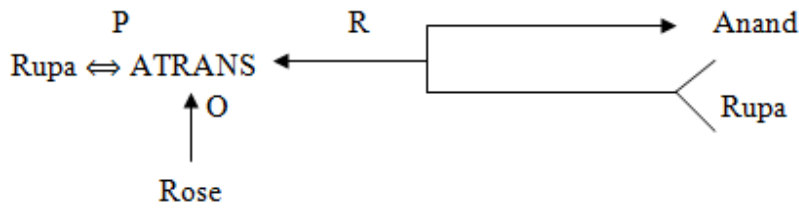
2) Bhabani cuts an apple with a knife



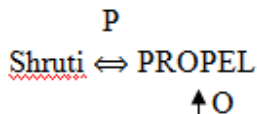
3) Sanjay drove the car fast



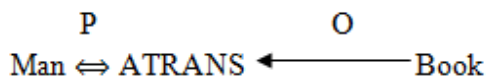
4) The rose was given by Rupa to Anand



5) Shruti pushed the door.

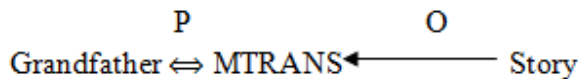


6) The man took a book

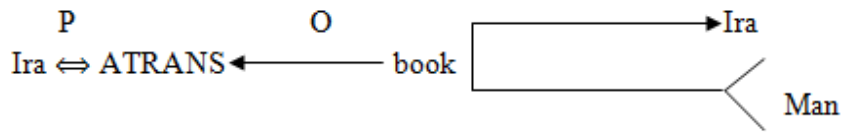


Here man is the doctor and book is the object of the action took.

7) My grandfather told me a story



8) Ira gave the man a dictionary



SCRIPT

It is another knowledge representation technique. Scripts are frame-like structures used to represent commonly occurring experiences such as going to a restaurant, visiting a doctor. 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. Scripts are useful because in the real world, there are no patterns to the occurrence of events. These patterns arise because of clausal relationships between events. The events described in a script form a giant causal chain. The beginning of the chain is the set of entry conditions which enable the first events of the script to occur. The end of the chain is the set of results which may enable later events to occur. The headers of a script can all serve as indicators that the script should be activated.

Once a script has been activated, there are a variety of ways in which it can be useful in interpreting a particular situation. A script has the ability to predict events that have not explicitly been observed. An important use of scripts is to provide a way of building a single coherent interpretation from a collection of observations. Scripts are less general structures than are frames and so are not suitable for representing all kinds of knowledge. Scripts are very useful for representing the specific kinds of knowledge for which they were designed.

A script has various components like:

- 1) **Entry condition:** It must be true before the events described in the script can occur. E.g. in a restaurant script the entry condition must be the customer should be hungry and the customer has money.
- 2) **Tracks:** It specifies particular positions in the script e.g. In a supermarket script the tracks may be cloth gallery, cosmetics gallery etc.
- 3) **Result:** It must be satisfied or true after the events described in the script have occurred. e.g. In a restaurant script the result must be true if the customer is pleased.

The customer has less money.

- 4) **Probs:** It describes the inactive or dead participants in the script e.g. In a supermarket script, the probes may be clothes, sticks, doors, tables, bills etc.

- 5) **Roles:** It specifies the various stages of the script. E.g. In a restaurant script the scenes may be entering, ordering etc.

Now let us look on a movie script description according to the above component.

- a) Script name : Movie
- b) Track : CINEMA HALL
- c) Roles : Customer(c), Ticket seller(TS), Ticket Checker(TC), Snacks Sellers (SS)
- d) Probes : Ticket, snacks, chair, money, Ticket, chart
- e) Entry condition : The customer has money
The customer has interest to watch movie.

6) **Scenes:**

a. SCENE-1 (Entering into the cinema hall)

- C PTRANS C into the cinema hall
- C ATTEND eyes towards the ticket counter
- C PTRANS C towards the ticket counters
- C ATTEND eyes to the ticket chart
- C MBUILD to take which class ticket
- C MTRANS TS for ticket
- C ATRANS money to TS
- TS ATRANS ticket to C

b. SCENE-2 (Entering into the main ticket check gate)

- C PTRANS C into the queue of the gate
- C ATRANS ticket to TC
- TC ATTEND eyes onto the ticket
- TC MBUILD to give permission to C for entering into the hall
- TC ATRANS ticket to C
- C PTRANS C into the picture hall.

c. SCENE-3 (Entering into the picture hall)

C ATTEND eyes into the chair
TC SPEAK where to sit
C PTRANS C towards the sitting position
C ATTEND eyes onto the screen

d. SCENE-4 (Ordering snacks)

C MTRANS SS for snacks
SS ATRANS snacks to C
C ATRANS money to SS
C INGEST snacks

e. SCENE-5 (Exit)

C ATTEND eyes onto the screen till the end of picture
C MBUILD when to go out of the hall
C PTRANS C out of the hall

7) Result:

The customer is happy

The customer has less money

Example 2: Write a script of visiting a doctor in a hospital

- 1) SCRIPT_NAME : Visiting a doctor
- 2) TRACKS : Ent specialist
- 3) ROLES : Attendant (A), Nurse(N), Chemist (C),
Gatekeeper(G), Counter clerk(CC), Receptionist(R), Patient(P),
Ent specialist Doctor (D), Medicine
Seller (M).
- 4) PROBES : Money, Prescription, Medicine, Sitting chair,
Doctor's table, Thermometer, Stetho scope, writing pad, pen,
torch, stature.

5) ENTRY CONDITION: The patient need consultation.

Doctor's visiting time on.

6) SCENES:

a. SCENE-1 (Entering into the hospital)

P PTRANS P into hospital

P ATTEND eyes towards ENT department

P PTRANS P into ENT department

P PTRANS P towards the sitting chair

b. SCENE-2 (Entering into the Doctor's Room)

P PTRANS P into doctor's room

P MTRANS P about the diseases

P SPEAK D about the disease

D MTRANS P for blood test, urine test

D ATRANS prescription to P

P PTRANS prescription to P.

P PTRANS P for blood and urine test

c. SCENE-3 (Entering into the Test Lab)

P PTRANS P into the test room

P ATRANS blood sample at collection room

P ATRANS urine sample at collection room

P ATRANS the examination reports

d. SCENE-4 (Entering to the Doctor's room with Test reports)

P ATRANS the report to D

D ATTEND eyes into the report

D MBUILD to give the medicines

D SPEAK details about the medicine to P

P ATRANS doctor's fee

P PTRANS from doctor's room

e. SCENE-5 (Entering towards medicine shop)

P PTRANS P towards medicine counter

P ATRANS Prescription to M

M ATTEND eyes into the prescription

M MBUILD which medicine to give

M ATRANS medicines to P

P ATRANS money to M

P PTRANS P from the medicine shop

7) RESULT:

The patient has less money

Patient has prescription and medicine.

Advantages And Disadvantages Of Different Knowledge Representation

Sl. No.	Scheme	Advantages	Disadvantages
1	Production rules	<ul style="list-style-type: none">• Simple syntax• Easy to understand• Simple interpreter• Highly Modular• Easy to add or modify	<ul style="list-style-type: none">• Hard to follow Hierarchies• Inefficient for large systems• Poor at representing structured descriptive knowledge.
2	Semantic	<ul style="list-style-type: none">• Easy to follow hierarchy• Easy to trace associations• Flexible	<ul style="list-style-type: none">• Meaning attached to nodes might be ambiguous• Exception handling is difficult• Difficult to program
3	Frame	<ul style="list-style-type: none">• Expressive Power• Easy to set up slots for new properties and relations• Easy to create specialized	<ul style="list-style-type: none">• Difficult to program• Difficult for inference• Lack of inexpensive software

		procedures	
4	Script	<ul style="list-style-type: none"> • Ability to predict events • A single coherent interpretation may be build up from a collection of observations 	<ul style="list-style-type: none"> • Less general than frames • May not be suitable to represent all kinds of knowledge
5	Formal Logic	<ul style="list-style-type: none"> • Facts asserted independently of use • Assurance that only valid consequence are asserted • Completeness 	<ul style="list-style-type: none"> • Separation of representation and processing • Inefficient with large data sets • Very slow with large knowledge bases

HUMAN ASSOCIATIVE MEMORY (HAM)

This model was developed by John Anderson and Gordon Bower (1973). This memory is organized as a network of propositional binary trees. When an informant asserts a statement to HAM, the system parses the sentence and builds a binary tree representation. As HAM is informed of new sentences, they are parsed and formed into new tree like structures with existing ones. When HAM is posed with a query it is formed into a tree structure called a probe. This structure is then matched against memory structures for the best match. The structure with the closest match is used to formulate an answer to the query. Matching is accomplished by first locating the leaf nodes in memory that match leaf nodes in the probe. The corresponding links are then checked to see if they have the same labels and in the same order. The search process is constrained by searching only node groups that have the same relation links. Access to nodes in HAM is accomplished through word indexing in LISP.

In HAM, nodes in the tree are assigned with unique numbers, while links are labeled with some functions. They are given below:

C : Context for free fact

E : Set membership

F : a fact

L : a location

O : An object

P : Predicate

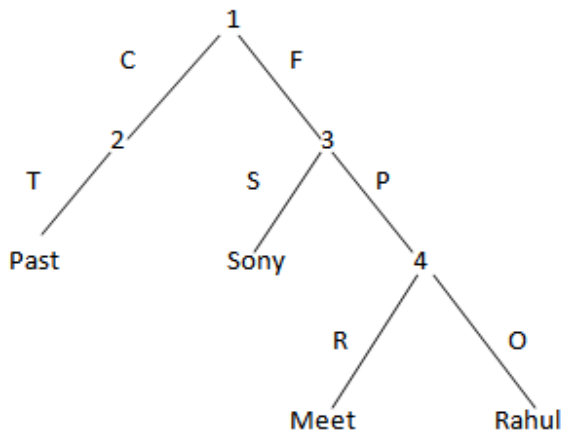
R : Relation

S : Subject

T : Time (Present, past, future)

On the basis of above function, we can represent various sentences in HAM. Let us look some example by using the above functions.

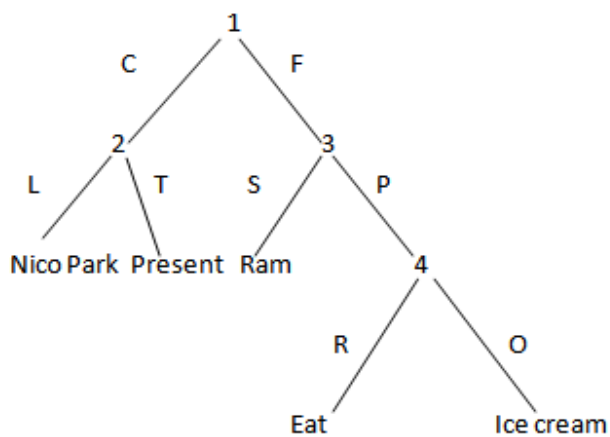
1) Sony met Rahul



Figure

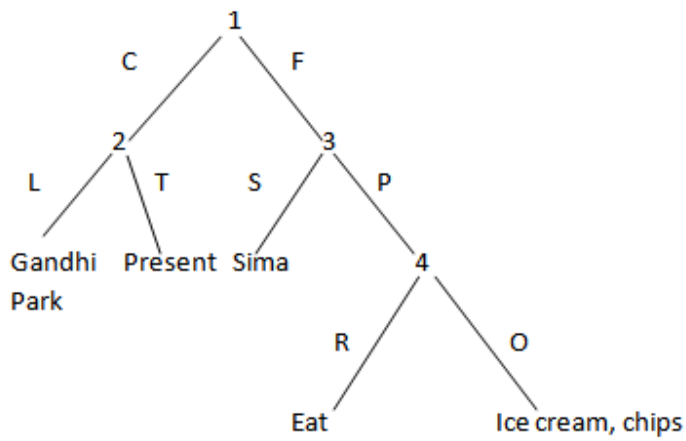
On the above Ham Structure the time is the past (met), as Sony did the work so Sony is the subject and Rahul will be the object and the relation is met.

2) Ram is eating an apple at Nico Park.



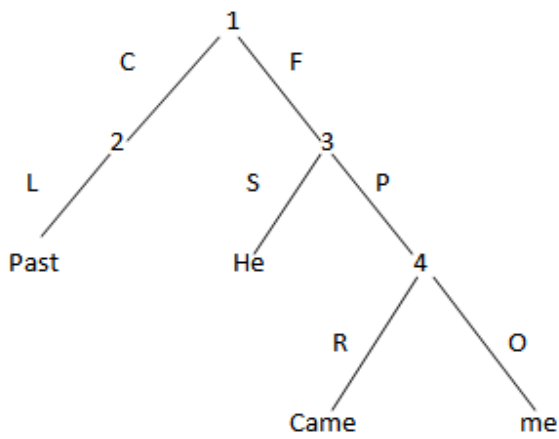
Figure

3) Sima is eating ice cream as well as chips at Gandhi Park.



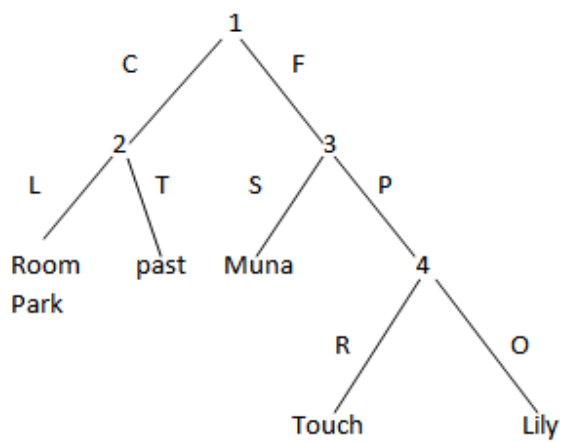
Figure

4) He came to me.



Figure

5) In a room Muna touched Lily.



Figure

Module 2

MIN-MAX Search

Games have always been an important application area for heuristic algorithms. In playing games whose state space may be exhaustively delineated, the primary difficulty is in accounting for the actions of the opponent. This can be handled easily by assuming that the opponent uses the same knowledge of the state space as us and applies that knowledge in a consistent effort to win the game. Minmax implements game search under referred to as MIN and MAX.

The min max search procedure is a depth first, depth limited search procedure. The idea is to start at the current position and use the plausible move generator to generate the set of possible successor positions. To decide one move, it explores the possibilities of winning by looking ahead to more than one step. This is called a ply. Thus in a two ply search, to decide the current move, game tree would be explored two levels farther.

Consider the below example

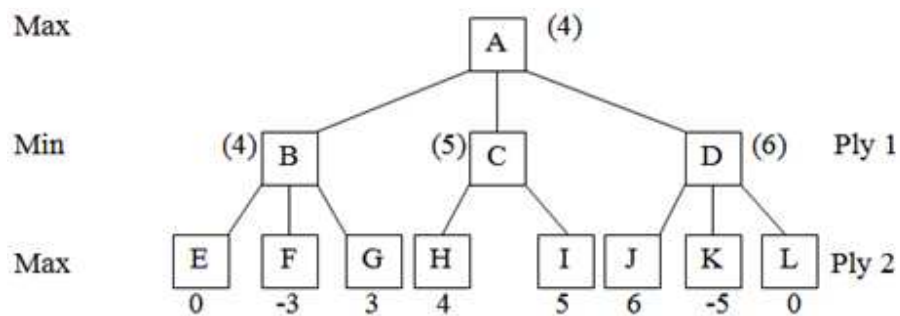


Figure Tree showing two ply search

In this tree, node A represents current state of any game and nodes B, C and D represent three possible valid moves from state A. similarly E, F, G represents possible moves from B, H, I from C and J, K, L, from D. to decide which move to be taken from A, the different possibilities are explored to two next steps. 0, -3, 3, 4, 5, 6, -5, 0 represent the utility values of respective move. They indicate goodness of a move. The utility value is back propagated to ancestor node, according to situation whether it is max ply or min ply. As it is a two player game, the utility value is alternatively maximized and minimized. Here as

the second player's move is maximizing, so maximum value of all children of one node will be back propagated to node. Thus, the nodes B, C, D, get the values 4, 5, 6 respectively. Again as ply 1 is minimizing, so the minimum value out of these i.e. 4 is propagated to A. then from A move will be taken to B.

MIN MAX procedure is straightforward recursive procedure that relies on two auxiliary procedures that are specific to the game being played.

1. MOVEGEN (position, player): the move generator which returns a list of nodes representing the moves that can be made by player in position. We may have 2 players namely PLAYER-TWO in a chess problem.
2. STATIC (position, player): the static evaluation function, which returns a number representing the goodness of position from the standpoint of player.

We assume that MIN MAX returns a structure containing both results and that we have two functions, VALUE and PATH that extract the separate components. A function LAST PLY is taken which is assumed to evaluate all of the factors and to return TRUE if the search should be stopped at the current level and FALSE otherwise.

MIN MAX procedure takes three parameters like a board position, a current depth of the search and the players to move. So the initial call to compute the best move from the position CURRENT should be

MIN MAX (CURRENT, 0, PLAYER-ONE)

(If player is to move)

Or

MIN MAX (CURRENT, 0, PLAYER-TWO)

(If player two is to move)

Let us follow the algorithm of MIN MAX

Algorithm: MINMAX (position, depth, player)

1. If LAST PLY (position, depth)
Then RETURN VALUE = STATIC (position, player)
PATH = nil.
2. Else, generate one more ply of the tree by calling the function MOVE_GEN (position, player) and set SUCCESSIONS to the list it returns.
3. If SUCCESSIONS is empty,

THEN no moves to be made
 RETURN the same structure that would have been returned if LAST_PLY had returned TRUE.

4. If SUCCESSIONS is not empty,
 THEN examine each element in turn and keep track of the best one.
5. After examining all the nodes,
 RETURN VALUE = BEST- SCORE
 PATH = BEST- PATH

When the initial call to MIN MAX returns, the best move from CURRENT is the first element in the PATH.

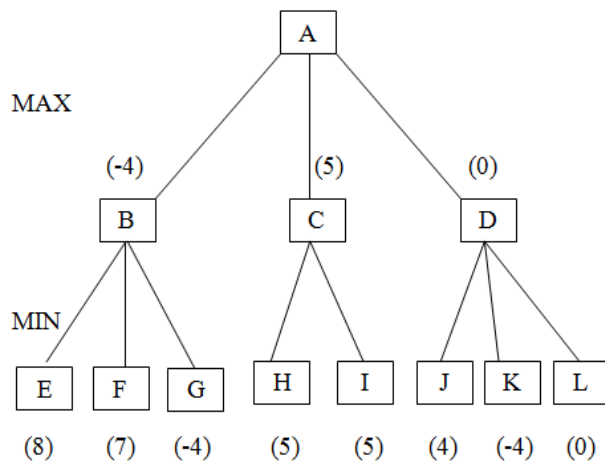
Alpha- Beta (α - β) Pruning

When a number of states of a game increase and it cannot be predicted about the states, then we can use the method pruning. Pruning is a method which is used to reduce the no. of states in a game. Alpha- beta is one such pruning technique. The problem with minmax search is that the number of game states it has to examine is exponential in the number of moves. Unfortunately we cannot eliminate the exponent, but we can effectively cut it in half. Alpha-beta pruning is one of the solutions to the problem of minmax search tree. When α - β pruning is applied to a standard minmax tree, it returns the same move as minmax would, but prunes away branches that cannot possibly influence the final decision.

The idea of alpha beta pruning is very simple. Alpha beta search proceeds in a depth first fashion rather than searching the entire space. Generally two values, called alpha and beta, are created during the search. The alpha value is associated with MAX nodes and the beta value is with MIN values. The value of alpha can never decrease; on the other hand the value of beta never increases. Suppose the alpha value of A MAX node is 5. The MAX node then need not consider any transmitted value less than or equal to 5 which is associated with any MIN node below it. Alpha is the worst that MAX can score given that MIN will also do its best. Similarly, if a MIN has a beta value of 5, it need not further consider any MAX node below it that has a value of 6 or more.

The general principal is that: consider a node η somewhere in the search tree, such that player has a choice of moving to that node. If player has a better choice K either at the parent node of η or at any choice point further up, then η will never be reached in actual play. So once we have found out enough about η (by examining some of its descendents) to reach this conclusion, we can prune it.

We can also say that " α " is the value of the best choice we have found so far at any choice point along the path for MAX. Similarly " β " is the value of the best choice we have found so far at any choice point along the path for MIN. Consider the following example



Figure

Here at MIN ply, the best value from three nodes is -4, 5, 0. These will be back propagated towards root and a maximizing move 5 will be taken. Now the node E has the value 8 is far more, then accepted as it is minimizing ply. So, further node E will not be explored. In the situation when more plies are considered, whole sub tree below E will be pruned. Similarly if $\alpha=0$, $\beta=7$, all the nodes and related sub trees having value less than 0 at maximizing ply and more than 7 at minimizing ply will be pruned.

Alpha beta search updates the value of α and β as it goes along and prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current α and β value for MAX or MIN respectively. The effectiveness of alpha- beta pruning is highly dependent on the order in which the successors are examined suppose in a search tree the branching factor is x and depth d . the α - β search needs examining only $x^{d/2}$ nodes to pick up best move, instead of x^d for MINMAX.

Constraint Satisfaction Search

A constraint search does not refer to any specific search algorithm but to a layer of complexity added to existing algorithms that limit the possible solution set. Heuristic and acquired knowledge can be combined to produce the desired result a constraint satisfaction problem is a special kind of search problem in which states are defined by the values of a set of variables and the goal state specifies a set of constraints that the value must obey. There are many problems in AI in which the goal state is not specified in the problem and it requires to be discovered according to some specific constraint. Examples of some constraint satisfaction search include design problem, labeling graphs, robot path planning and cryptarithmic problem etc.

A constraint satisfaction problem (CSP) is defined by a set of variables (x_1, x_2, \dots, x_n) and a set of constraints (c_1, c_2, \dots, c_m) . Each variable x_i has a non empty domain d_i of possible values. Each constraint c_i in values some subset of the variables and specifies the allowable combination of values for

that subset. The search space of CSPs is often exponential. Therefore a number of different approaches to the problem have been proposed to reduce the search space and find a feasible solution in a reasonable time based on the search space exploring and variable selection heuristics different algorithms and can be developed for a CSP problem. The algorithms can be divided into two major categories such as complete and incomplete algorithm.

Complete algorithms seek any solution or solutions of a CSP or they try to prove that no solution into the categories like constraint propagation techniques which tries to eliminate values that are consistent with some constraints and systematic search techniques. Which explores systematically the whole search space. But on the other hand incomplete search methods do not explore the whole search space. They search the space either non-systematically or in a systematic manner, but with a limit on some resource.

They may not provide a solution but their computational time is reasonably reduced. They cannot be applied to find all solutions or to prove that no solution exists. Let us look an algorithm to solve a constraint satisfaction problem.

Algorithm:

- 1) Open all objects that must be assigned values in a complete solution.
- 2) Repeat until all objects assigned valid values.
- 3) Select an object and strengthen as much as possible. The set of constraints that apply to object.
- 4) If set of constraints is different from previous set then open all objects that share any of these constraints. Remove selected objects.
- 5) If union of constraints discovered above defines a solution, return solution.
- 6) If union of constraints discovered above defines a contradiction, return failure.
- 7) Make a guess in order to proceed. Repeat until a solution is found.
- 8) Select an object with a number assigned value and try strengthen its constraints.

PLANNING

The process of doing a sequence of actions to achieve a goal is called planning. A plan is a representation of the crude structure of the input scene by the various object labels. The process of planning is a bottom up process to provide clues concerning which knowledge can be applied to different parts of the scene. The knowledge of the task world is represented by sets of productions rules. Each rule in the bottom up process has a fuzzy predicate which describes the properties of relations between objects. Generally there are various agents who act to plan. The environments for an agent may be deterministic, finite, static in which change happens only when the agent acts. The discrete environment includes the time factor, objects, effects etc. These environments are called classical planning environments. On the other hand, the non classical planning environments are partially observable and involves a different set of algorithms

and agent designs. Planning refers to the process of computing several steps of a problem solving procedure before evaluation of that problem.

Computer cannot solve any problem without planning it. For example, in 8-puzzle game, the computer can't replace the tiles onto their positions without the planning procedure of that problem. When we discuss the computer solution of the 8-puzzle game, what we are really doing was outlining the way the computer might generate a plan for solving it. A computer could look for a solution plan in the same way as a person who was actually trying to solve the problem by moving tiles on a board. If solution steps in the real world cannot be ignored or undone, though planning becomes extremely important. Although real world steps may be irrevocable, computer simulation of those steps is not. So we can circumvent the constraints of the real world by looking for a complete solution in a simulated world in which backtracking is allowed. After we find a solution, we can execute it in the real world. The fact that we can leave out properties of world states that are irrelevant to the problem at hand or that are not known is one of the powerful aspects of using a feature based approach. This aspect is particularly important in describing the goal condition that we want the agent to achieve by its actions.

Basic Components of a Planning System

When a particular problem will be solved, at that time some specific rules regarding to that problem are to be applied. Then apply the chosen rule to compute the new problem state that arises from its application. Detect when a solution has been found and calculate the active and inactive ends of that problem. Various components of a planning system are described as follows.

- (a) **States:** For a planning process, the planners decompose the world into some environments. Then environments are defined by some logical conditions and states. The problems can be viewed as the task of finding a path from a given starting state to some desirable goal state. The state can be viewed as a conjunction of positive literals. For example, Rich A famous might represent the state of a best agent.
- (b) **Goal:** A goal is a specified state. To find a solution to a problem using a search procedure is to generate moves through the problem space until a goal state is reached. In the context of game playing programs, a goal state is one in which we win. Unfortunately, for interesting games like chess, it is not usually, possible, even with a good plausible move generator, to search until a goal state is found.
- (c) **Actions:** An action is specified in terms of the pre-conditions that must hold before it can be executed and then the effects that ensue when it is executed. For example, an action for running a tiger from one location to another is

Action (Run (T, from, to),

PRECONDITION: $At(T, from) \wedge Tiger(T) \wedge Jungle(from) \wedge Jungle(To)$

EFFECT: $\sim \text{At}(T, \text{from}) \wedge \text{At}(T, \text{to})$)

- (d) **Precondition:** The precondition is a conjunction of function free positive literals stating what must be true in a state before the action can be executed.
- (e) **Effect:** It is a conjunction of function free literals describing how the state changes when the action is executed.
- (f) **Finding a solution:** A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transforms the initial problem state into the goal state. The way it can be solved depends on the way that state descriptions are represented.
- (g) **Calculating the Dead State:** As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution. The same reasoning methods that can be used to detect a solution can often be used for detecting a dead path. If the search process is reasoning in forward direction from the initial state, it can prune any path that leads to a state from which the goal state cannot be reached. If the search process is reasoning backward from the goal state, it can also terminate a path either because it is sure that the starting state cannot be reached.

Planning in State Space Search

Problem solving in AI may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. The problem solving agents decide what to do by finding sequence of action that lead to desirable states. The simplest agents which have been described below are the reflex and goal based agents. The reflex agents use direct mapping from states to actions and are unsuitable for very large mappings. Problem solving agents find action sequence that lead to desirable state.

A state space is represented by four components like steps involved in a problem solving process, the start state of the problem and the corresponding goal state. Search algorithms should track the paths from the start node to the goal node because these paths contain a series of operations that lead to the solution of the problem. A programmer should analyze and predict the behaviour of search algorithms to successfully designed and implement them in a proper manner. The problems can be characterized as spaces consisting of a set of states and a set of operators that map from one state to another state. The states may be distinguished as containing one of the following: one or more initial or starting states, a number of intermediate states and one or more goal states. A sequence of operators that map an initial state to a goal state will provide the solution to a problem. A best solution is one that requires the fewest no. of operations while mapping from an initial state to the goal state. The amount of time and memory space required to complete the mapping measures the performance of a particular solution method. The state space search can be in forward and backward direction. The forward state space planning is also known as

progression planning in which searching always takes place in forward direction. In backward search, it finds only the relevant actions. An agent with several immediate options of unknown values can decide what to do by first examining the different possible sequences of actions that lead to states of known values and then choosing the best one.

A state space search can be searched in two directions like from the inputs towards the goal or from the goals towards the inputs. In data driven search, one starts with the given facts of a problem and uses a set of legal moves or rules to change the states. This process is continued until it generates a path that satisfies the goal condition. In goal driven search, first determine the rules or legal moves that can be used to generate the goal and identifies the condition that can be applied to use these rules. These conditions form the new goals or sub goals for the search. One must continue the search process by working backwards through successive sub goals until it returns of moves or rules leading from the data to a goal, even if it performs this process backwards. Data driven search is suggested if one is provided with almost all the data at the time of formulation of the problem statement.

Data driven search uses the knowledge and constraints present in the given data of a problem to guide the search along a path. The main work in the area of search strategies is to find the correct search performance measures like time complexity, space complexity, completeness and optimality help to judge the fitness of a particular search algorithm.

Various Planning Techniques

Several planning techniques are described below.

- (1) **Hierarchical Planning:** In hierarchical planning, at each level of hierarchy the objective functions are reduced to a small number of activities at the next lower level. So the computational cost of finding the correct way to arrange these activities for the current problem is small. Hierarchical methods can result in linear time. The initial plan of hierarchical planning describes the complete problem which is a very high level description. The plans are refined by applying action decompositions. Each action decomposition reduces a high level description to some of the individual lower level descriptions. The action decomposers describe how to implement the actions.
- (2) **Conditional Planning:** It deals with the planning by some appropriate conditions. The agents plan first and then execute the plan that was produced. The agents find out which part of the plan to execute by including sensing actions in the plan to test for the appropriate conditions.
- (3) **Exact Planning:** It is also called as conformation planning. It ensures that the plan achieves the goal in all possible circumstances regardless of the true initial state and the actual actions outcome. This planning is based on the idea that the world can be forced into a given state even when the agent has only partial information about the current state.

- (4) **Replanning:** It occurs when there is any wrong information regarding with the planning. The agent can plan the same plan as the conditional planner or some new steps.
- (5) **Continuous Planning:** In this planning, the planner at first achieves the goal and then only can stop. A continuous planner is designed to persist over a lifetime. It can handle any unfavorable circumstances in the environment.
- (6) **Multiagent Planning:** In multiagent planning some other new agents may involved with our single agent in the environment. This may lead to a poor performance because dealing with other agents is not the same as dealing with the nature. It is necessary when there are other agents in the environment with which to cooperate, compete or coordinate.
- (7) **Multibody Planning:** This planning constructs joint plans, using an efficient decomposition of joint action descriptions, but must be augmented with some form of co-ordination of two cooperative agents are to agree on which joint plan to execute.

UNDERSTANDING

Understanding is the simplest procedure of all human beings. Understanding means ability to determine some new knowledge from a given knowledge. For each action of a problem, the mapping of some new actions is very necessary. Mapping the knowledge means transferring the knowledge from one representation to another representation. For example, if you will say “I need to go to New Delhi” for which you will book the tickets. The system will have “understood” if it finds the first available plane to New Delhi. But if you will say the same thing to you friends, who knows that your family lives in “New Delhi”, he/she will have “understood” if he/she realizes that there may be a problem or occasion in your family. For people, understanding applies to inputs from all the senses. Computer understanding has so far been applied primarily to images, speech and typed languages. It is important to keep in mind that the success or failure of an “understanding” problem can rarely be measured in an absolute sense but must instead be measured with respect to a particular task to be performed. There are some factors that contribute to the difficulty of an understanding problem.

- (a) If the target representation is very complex for which you cannot map from the original representation.
- (b) There are different types of mapping factors may arise like one-to-one, one-to-many and many-to-many.
- (c) Some noise or disturbing factors are also there.
- (d) The level of interaction of the source components may be complex one.
- (e) The problem solver might be unknown about some more complex problems.
- (f) The intermediary actions may also be unavailable.

Consider an example of an English sentence which is being used for communication with a keyword-based data retrieval system. Suppose I want to know all about the temples in India. So I would need to be translated into a representation such as

(SEARCH KEYWORDS = TEMPLE &INDIA)

The above sentence is a simple sentence for which the corresponding representation may be easy to implement. But what for the complex queries?

Consider the following query.

“Ram told Sita he would not eat apple with her. He has to go to the office”. This type of complex queries can be modeled with the conceptual dependency representation which is more complex than that of simple representation. Constructing these queries is very difficult since more information are to be extracted. Extracting more information will require some more knowledge.

Also the type of mapping process is not quite easy to the problem solver. Understanding is the process of mapping an input from its original form to a more useful one. The simplest kind of mapping is “one-to-one”. In one-to-one mapping each different problems would lead to only one solution. But there are very few inputs which are one-to-one. Other mappings are quite difficult to implement. Many-to-one mappings are frequent is that free variation is often allowed, either because of the physical limitations of that produces the inputs or because such variation simply makes the task of generating the inputs. Many-to-one mapping require that the understanding system know about all the ways that a target representation can be expressed in the source language. One-to-many mapping requires a great deal of domain knowledge in order to make the correct choice among the available target representation. The mapping process is simplest if each component can be mapped without concern for the other components of the statement. If the number of interactions increases, then the complexity of the problem will increase. In many understanding situations the input to which meaning should be assigned is not always the input that is presented to the under stander. Because of the complex environment in which understanding usually occurs, other things often interfere with the basic input before it reaches the under stander. Hence the understanding will be more complex if there will be some sort of noise on the inputs.

NATURAL LANGUAGE PROCESSING

Natural language processing is a subfield of computer science and in artificial intelligence that is concerned with computational processing of natural languages, emulating cognitive capabilities without being committed to a true simulation of cognitive processes. It is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human like language processing for a range of tasks or applications. It is a computerized approach to analyzing text that is based on both a set of theories and a set of technologies. NLP is a very active area of research and development. Naturally occurring texts can be of any language, mode and genre etc. The text can be oral or written. The only requirement is that they be in a language used by humans to communicate to one another. Also, the text being analyzed should not

be specifically constructed for the purpose of analysis, but rather that the text is gathered from actual usage.

The notion of levels of linguistic analysis refers to the fact that there are multiple types of language processing known to be at work when humans produce or comprehend language. The humans use generally various types of sentences for expressing their feelings. Sentences are classified by structure and usage. A simple sentence has one independent clause comprised of a subject and predicate. A compound sentence consists of two or more independent clauses connected by a conjunction or a semicolon. The way a sentence is used determines its mood, declarative, imperative, interrogative or exclamatory. A word functions in a sentence as a part of speech. Parts of speech for the English language are nouns, pronouns, verbs, adjectives, adverbs, prepositions, conjunctions etc.

Generally NLP is the means for accomplishing a particular task. It is a combination of computational linguistics and artificial intelligence. The natural language processing uses the tools of AI such as: algorithms, data structures, formal models for representing knowledge, models or reasoning processes etc. There are two ways through which the natural languages are being processed. First parsing technique and the second is the transition network. The architecture of NLP is given figure .

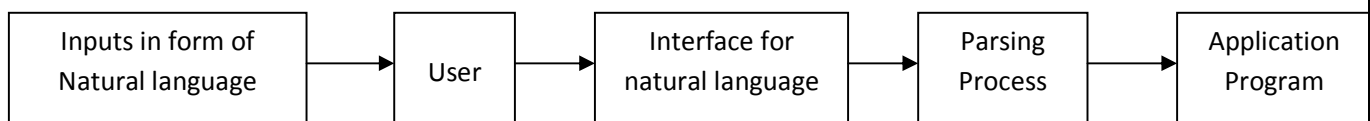


Figure Architecture of a NLP

In NLP, to interact with the database in natural languages, computer is required to have knowledge of basic alphabet, lexicon, grammar and words formation etc. The inputs are in the form of natural language given by the user. Finally after parsing process the output in the language is being understood by the application program.

GOALS OF NLP

The goal of natural language processing is to specify a language comprehension and production theory to such a level of detail that a person is able to write a computer program which can understand and produce natural language. The basic goal of NLP is to accomplish human like language processing. The choice of word “processing” is very deliberate and should not be replaced with “understanding”. For although the field of NLP was originally referred to as Natural Language Understanding (NLU), that goal has not yet been accomplished. A full NLU system would be able to:

→ Paraphrase an input text.

- Translate the text into another language.
- Answer questions about the contents of the text.
- Draw inferences from the text.

While NLP has made serious inroads into accomplishing goals from first to third, the fact that NLP system can not, of themselves, draw inferences from text, NLU still remains the goal of NLP. Also there are some practical applications of NLP. An NLP-based IR system has the goal of providing more precise, complete information in response to a user's real information need. The goal of the NLP system is to represent the true meaning and intent of the user's query, which can be expressed as naturally in everyday language.

APPLICATIONS OF NLP

NLP lie in a number of disciplines like computer and information sciences, linguistics, mathematics, electrical and electronic engineering, artificial intelligence and robotics, psychology etc. Applications of NLP include a number of fields of studies such as machine translation, natural language text processing, summarization, user interfaces multilingual and Cross language information retrieval (CLIR), speech recognition, artificial intelligence and expert system. Research on NLP is regularly published in a number of conferences such as the annual proceedings of ACL (Association of Computational Linguistics) and its European counter part EACL, biennial proceedings of the Message Understanding Conferences (MUCS), Text Retrieval Conferences (TRECS) and ACM-SIGIR (Association of Computing Machinery-Special Interest Group on Information Retrieval) conferences.

As natural language processing technology matures, it is increasingly being used to support other computer applications. Such use naturally falls into two areas, one in which linguistic analysis merely serves as an interface to the primary program and the second one in which natural language considerations are central to the application. Natural language interfaces into a request in a formal database query language, and the program then proceeds as it would without the use of natural language processing techniques. The design of question answering systems is similar to that for interfaces to database management systems. One difference however, is that the knowledge base supporting the question answering system does not have the structure of a database. Similarly in message understanding systems, a fairly complete linguistic analysis may be required but the messages are relatively short and the domain is often limited. Also some more application areas include information and text categorization. In both applications, natural language processing imposes a linguistic representation on each document being considered. In text categorization a collection of documents is inspected and all documents are grouped into several categories based on the characteristics of the linguistic representations of the documents. In information filtering documents satisfying some criterion are singled out from a collection.

Discourse Knowledge

While syntax and semantics work with sentence-length units, the discourse level of NLP works with units of text longer than a sentence i.e. it does not interpret multi-sentence texts as just concatenated sentences, each of which can be interpreted singly. Discourse focuses on the properties of the text as a whole that convey meaning by making connections between component sentences. Several types of discourse processing can occur at this level like anaphora resolution and discourse/text structure recognition. Anaphora resolution is the replacing of words such as pronouns which are semantically vacant with the appropriate entity to which they refer. For example, newspaper articles can be deconstructed into discourse components such as: lead, main story, previous events, evaluation etc. A discourse is a sequence of sentences. Discourse has structure much like sentences do. Understanding discourse structure is extremely important for dialog system.

For example: The dialog may be

When does the bus to Bhubaneswar leave?

There is one at 10 a.m. and one at 1 p.m.

Give me two tickets for the earlier one, please.

The problems with discourse analysis may be non-sentential utterances, cross-sentential anaphora.

Pragmatic Knowledge

This level is concerned with the purposeful use of language in situations and utilizes context over and above the contents of the text for understanding. The goal is to explain how extra meaning is read into texts without actually being encoded in them. This requires much world knowledge including the understanding of intentions, plans and goals. Some NLP applications may utilize knowledge bases and inferencing modules. Pragmatic is the study of how more gets communicated than is said. Speech acts in the pragmatic processing is the illocutionary force, the communicative force of an utterance, resulting from the function associated with it. For example: Suppose the sentence is I will see you later.

Prediction: I predict that I will see you later.

Promise: I promise that I will see you later.

Warning: I warn you that I will see you later.

EXAMPLES OF SOME NLP SYSTEMS

In early 1950s, few NLP systems had been developed. Except the theoretical developments many practical systems were developed to demonstrate the effectiveness of particular principles. Weizenbaum's ELIZA was built to replicate the conversation between a psychologist and a patient; simply by permuting the user input. Winograd's SHRDLU simulated a robot that manipulated blocks on a table top. Also LUNAR was developed by Woods as an interface system to a database. In the late 1970's, McKeown's discourse planner TEXT and McDonald's response generator MUMMBLE used theoretical predicates to produce declarative descriptions in the form of short texts, usually paragraphs. Some of the earliest NLP systems are described below.

ELIZA System

ELIZA was an early natural language processing system capable of carrying on a limited form of conversation with a user. It was developed by Weizenbaum in 1966. It is a remarkably simple program that makes use of pattern matching to process the input and translate it into suitable outputs. Consider the following session with a version of ELIZA that limited the responses of a psychotherapist.

User 1:

You are like my brother in some ways.

ELIZA 1:

WHAT RESEMBLANCE DO YOU SEE?

User 2:

You are not very intelligent but I think you don't want me to notice that.

ELIZA 2:

WHAT MAKES YOU THINK I AM NOT INTELLIGENT?

User 3:

You don't agree with me.

ELIZA 3:

WHY DO YOU THINK I DON'T AGREE WITH YOU?

User 4:

You are afraid of me.

ELIZA 4:

DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU?

User 5:

I need some help, that much seems certain.

ELIZA 5:

WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP?

This program is one of the most popular AI programs of all time since it carried the keyword based analysis approach to an absurd (but impressive) extreme. The success of this simple technique in this domain is due to the fact that ELIZA doesn't actually need to know anything to mimic a psychotherapist. As Weizenbaum notes, this is one of the few dialogue genres where the listener can act if they know nothing of the world. ELIZA's deep relevance to Turing's ideas is that many people who interacted with ELIZA came to believe that it really understood them and their problems. Indeed, Weizenbaum (1976) notes that many of these people continued to believe in ELIZA's abilities even after the program's operation was explained to them.

LUNAR System

It was developed by Woods in 1970. It is one of the largest and most successful question-answering systems using AI techniques. This system had a separate syntax analyzer and a semantic interpreter. Its parser was written in ATN (Augmented Transition Network) form. The system was used in various tests and responded successfully to queries like followings:

- How many oak trees have height greater than 15 inches?
- What is the average concentration of hydrogen and oxygen in water?
- Which one is the oldest material between Iron, Bauxite and Aluminum?

The LUNAR system is mainly deal with queries. But the performance of the system is very good than other systems.

HAL System

HAL is an artificial agent capable of such advanced language processing behaviour as speaking and understanding English. The HAL system was developed by Arthur C. Clarke. Generally HAL system is useful for language and speech recognition. By speech and language processing we have in mind those computational techniques that process spoken and written human language. HAL require much broader and deeper knowledge of language. To determine what the user is saying, HAL must be capable of analyzing an incoming audio signal and recovering the exact sequence of words user used to produce that

signal. Similarly, in generating its response HAL must be able to take a sequence of words and generate an audio signal that the user can recognize. Both of these tasks require knowledge about phonetics and phonology which can help model how words are pronounced in colloquial speech.

SHRDLU System

It was developed by Winograd in 1970. It was a dialogue system which could converse with a human user about simple world containing building blocks. It is a simulation based programming system involving of a hand and eye. It is a syntax based system which is a combination of deep and surface structure. It contains a syntactic parser with a fairly wide coverage which builds surface structures that are not simply of trivial category labeling. It performs the combination and integration of many components which will create a total system. For example: User 1: Chose a green pen.

SHRDLU: Ok

User 2: Write with the pen.

SHRDLU: I DON'T UNDERSTAND WHICH PEN YOU MEAN.

The Chomsky Hierarchy of Grammars

A hierarchy of classes of languages viewed as sets of strings, ordered by their “complexity”. The higher the language is in the hierarchy, the more complex it is. In particular the class of languages in one class properly includes the languages in lower classes. There exists a correspondence between the class of languages and the format of phrase structure rules necessary for generating all its languages. Noam Chomsky defined a hierarchy of grammars called Type 0, Type 1, Type 2 and Type 3. The outline of Chomsky hierarchy of languages is given in figure .

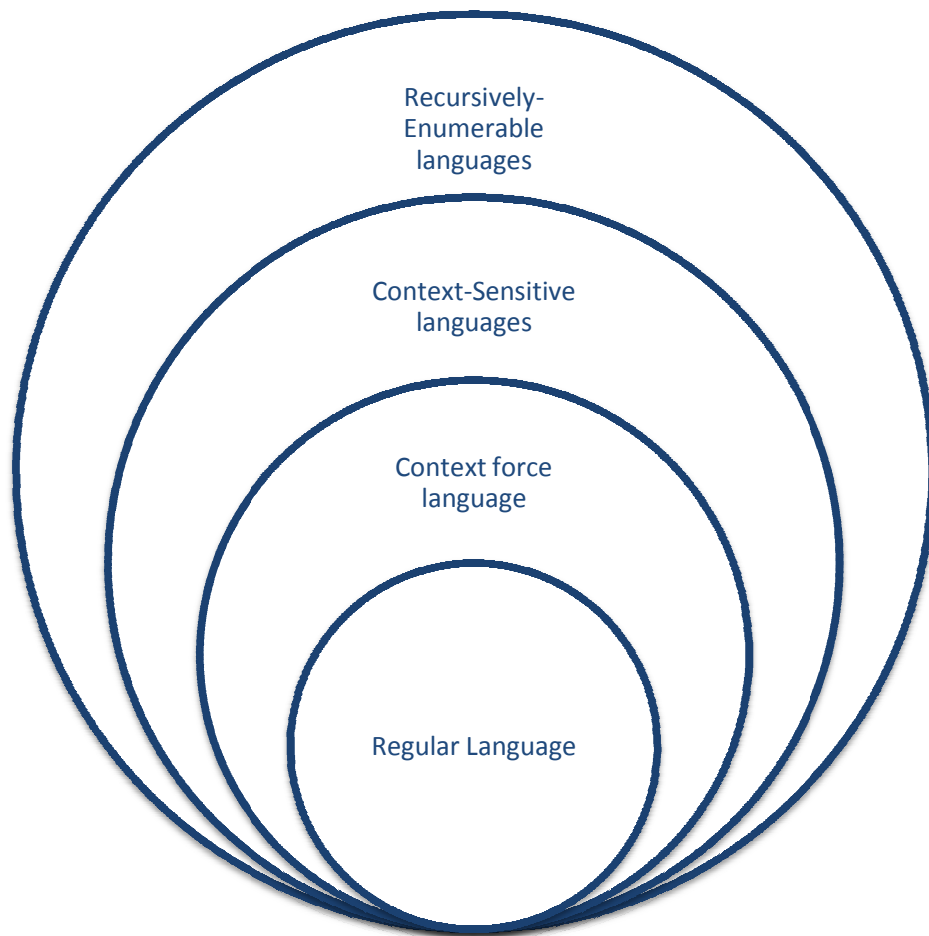


Figure Chomsky Hierarchy of Languages

The Chomsky hierarchy of languages reflects a certain order of complexity in some sense, the lower the language class is in the hierarchy; the simpler are its possible constructions.

1. **Type 0 (Recursively Enumerable Languages):** It is obtained by making the simple restrictions that cannot be empty string in the rewrite form $abc \rightarrow adc$. It is the much more simplest grammar than others.
2. **Type 1 (Context Sensitive Languages):** They have added restriction that the length of the string on the right hand side of the rewrite rule must be at least as long as the strings on the left side. In production of the form in $abc \rightarrow adc$, b must be a single non terminal symbol and d is a non empty string.
For example:

$$aA \rightarrow ab$$
$$aA \rightarrow aa$$
$$ba^* \rightarrow AB$$

3. Type 2 (Context Free Grammar): This type can be represented as

$\langle \text{Symbols} \rangle \rightarrow \text{Symbols } 1 \langle \text{Symbols } 2 \rangle \dots \langle \text{Symbols } n \rangle$ where $n \geq 1$. Also in type 2 grammars the left hand side is a single non terminal symbol which is very important. Its computational device is push down automata.

For example: $S \rightarrow as$

$$S \rightarrow asa$$
$$S \rightarrow aA$$
$$S \rightarrow aAB$$
$$A \rightarrow a$$
$$B \rightarrow b$$

4. Type 3 (Regular Languages): It is the most restrictive type grammar than others. The rules of type 3 grammar is $A \rightarrow aB$

$$A \rightarrow a.$$

Its computational device is finite state automata. Example: $B \rightarrow bA$

$$B \rightarrow ba$$

So we can say that always a terminal symbol should start first (like 'b') then any no. of symbols may come.

Some symbols are used to represent the grammars and they are described as follows.

- Sentences $\rightarrow S$
- Verb Phrase $\rightarrow VP$
- Noun Phrase $\rightarrow NP$
- Preposition Phrase $\rightarrow PP$
- Auxiliary $\rightarrow AUX$
- Preposition $\rightarrow PREP$
- Adverb $\rightarrow ADV$
- Adjectives $\rightarrow ADJ$
- Determiners $\rightarrow DEJ$

- Article → ART
- Noun → N
- Verb → V and so on.

Also while representing a sentence we have to follow some rules like as follows.

1. $S \rightarrow NP \ VP$
Or
 $S \rightarrow NP \ VP \ PP$
2. $VP \rightarrow AUX \ VP$
 $VP \rightarrow V \ NP$
 $VP \rightarrow V \ PP$
 $VP \rightarrow V \ ADJ$
 $VP \rightarrow V \ NP \ PP$
 $VP \rightarrow V$
 $VP \rightarrow AUX \ V \ NP$
3. $NP \rightarrow NP \ ART$
 $NP \rightarrow N \ ADJ$
 $NP \rightarrow N \ PP$
 $NP \rightarrow N$
4. $PP \rightarrow PREP \ NP$

By taking into consideration the above rules any types of syntactic grammars are possible. Always you have to remember a sentence must have at least two clauses like verb phrase (VP) and noun phrase (NP). Now let us see some other types of grammars used in language processing like transformational grammar, case grammar, systemic grammar and semantic grammar etc.

Transformational Grammar

These are the grammars in which the sentence can be represented structurally into two stages. Obtaining different structures from sentences having the same meaning is undesirable in language understanding systems. Sentences with the same meaning should always correspond to the same internal knowledge structures. In one stage the basic structure of the sentence is analyzed to determine the grammatical constituent parts and in the second stage just the vice versa of the first one. This reveals the surface structure of the sentence, the way the sentence is used in speech or in writing. Alternatively, we can also say that application of the transformation rules can produce a change from passive voice to active voice and vice versa. Let us see the structure of a sentence as given below.

1. Ram is eating an apple (In Active Voice)

Figure (Transformational grammar Tree representation of Active voice to sentence formation)

2. An apple is being eaten by Ram (In Passive Voice)

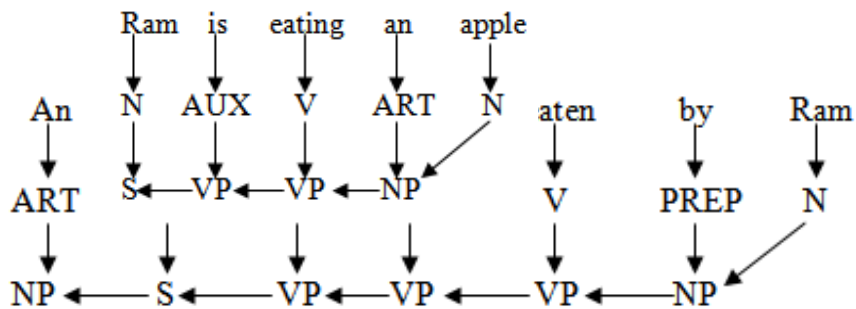


Figure Passive voice to sentence formation

Both of the above sentences are two different sentences but they have same meaning. Thus it is an example of a transformational grammar. These grammars were never widely used in computational models of natural language. The applications of this grammar are changing of voice (Active to Passive and Passive to Active) change a question to declarative form etc.

Case Grammars (FILLMORE's Grammar)

Case grammars use the functional relationships between noun phrases and verbs to conduct the more deeper case of a sentence. Generally in our English sentences, the difference between different forms of a sentence is quite negligible. In early 1970's Fillmore gave some idea about different cases of a English sentence. He extended the transformational grammars of Chomsky by focusing more on the semantic aspects of view of a sentence. In case grammars a sentence id defined as being composed of a preposition P, a modality constituent M, composed of mood, tense, aspect, negation and so on. Thus we can represent a sentence like

$$S \rightarrow M + P$$

Where P □ Set of relationships among verbs and noun phrases i.e. P = (C=Case)

M □ Modality constituent

For example consider a sentence "Ram did not eat the apple".

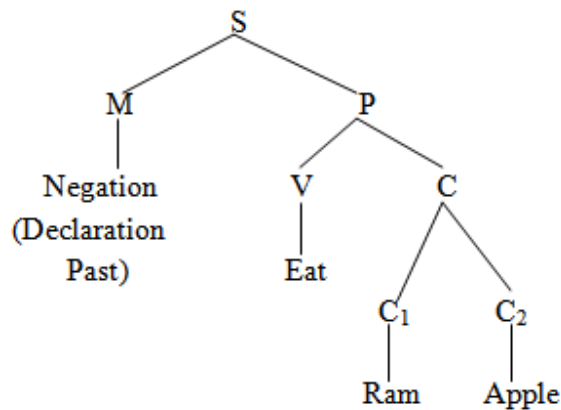


Figure Case Grammar Tree Representation

The tree representation for a case grammar will identify the words by their modality and case. The cases may be related to the actions performed by the agents, the location and direction of actions. The cases may also be instrumental and objective. For example “Ram cuts the apple by a knife”. Here knife is an instrumental case. In fig 8.5 the modality constituent is the negation part, eat is the verb and Ram, apple are nouns which are under the case C_1 and C_2 respectively. Case frames are provided for verbs to identify allowable cases. They give the relationships which are required and which are optional.

Semantic Grammars

Semantic grammars encode semantic information into a syntactic grammar. They use context free rewrite rules with non terminal semantic constituents. Generally a semantic error occurs when the meaning of the knowledge is not properly communicated. Semantic errors occur if the human expert misinterprets the knowledge engineer’s question or answers inappropriately. The proper constituents of semantic grammar are noun phrase (NP), verb phrase (VP), Noun (N), Verb (V), Preposition phrase (PP), Adverb (ADV) and so on. One of the successful applications of semantic grammar is the LIFER system (A database query system). In the LIFER system, there are several rules to handle the wh-queries such as:

What is the name of your country?

Which department of your country is most efficient?

Who guide them?

Where they are being guided?

These sentences are analyzed and words matched to the symbols contained in the lexicon entries. Semantic grammars are suitable for use in systems with restricted grammars since its power of computation is limited.

Context Free Grammar (CFG)

The grammar in which each production has exactly one terminal symbol in its left hand side and at least one symbol at the right hand side is called context free grammar. A CFG is a four tuple (Σ, V, S, P) where

Σ : Finite non empty set of terminals, the alphabet.

V : Finite non empty set of grammar variables (categories or non terminal symbols)

Such as $\Sigma \cap V = \emptyset$

S : Starting symbol ($S \in V$)

P : Finite set of production rules, each of the form $A \rightarrow \alpha$, where

$$A \in V \text{ and } \alpha \in (V \cup \Sigma)^*$$

Each terminal symbol in a grammar denotes a language. The non terminals are written in capital letters and terminals are written in small letters. Some properties of CFG formalism are

- Concatenation is the only string combination operation.
- Phrase structure is the only syntactic relationship.
- The terminal symbols have no properties.
- Non terminal symbols are atomic.
- Most of the information encoded in a grammar lies in the production rules.
- Any attempt of extending the grammar with semantics requires extra means.
- Concatenation is not necessarily the only way by which phrases may be combined to yield other phrases.
- Even if concatenation is the sole string operation, other syntactic relationships are being put forward.

For example we can write the followings:

```
S → NP VP
NP → ART N
VP → V NP
ART → a
N → Cat
N → Meat
V → Eat
```

PARSING PROCESS

Parsing is the term used to describe the process of automatically building syntactic analysis of a sentence in terms of a given grammar and lexicon. The resulting syntactic analysis may be used as input to a process of semantic interpretation. Occasionally, parsing is also used to include both syntactic and semantic analysis. The parsing process is done by the parser. The parser performs grouping and labeling of parts of a sentence in a way that displays their relationships to each other in a proper way.

The parser is a computer program which accepts the natural language sentence as input and generates an output structure suitable for analysis. The lexicon is a dictionary of words where each word contains some syntactic, some semantic and possibly some pragmatic information. The entry in the lexicon will contain a root word and its various derivatives. The information in the lexicon is needed to help determine the function and meanings of the words in a sentence. The basic parsing technique is shown in figure .

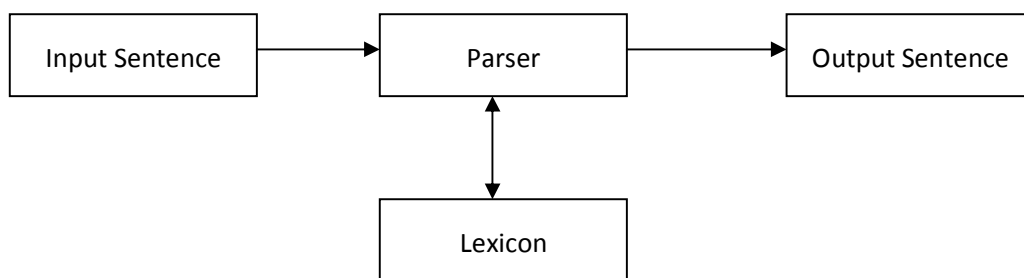


Figure Parsing Technique

Generally in computational linguistics the lexicon supplies paradigmatic information about words including part of speech labels, irregular plurals and sub categorization information for verbs. Traditionally, lexicons were quite small and were constructed largely by hand. The additional information being added to the lexicon increase the complexity of the lexicon. The organization and entries of a lexicon will vary from one implementation to another but they are usually made up of variable length data structures such as lists or records arranged in alphabetical order. The word order may also be given in terms of usage frequency so that frequently used words like “a”, “the” and “an” will appear at the beginning of the list facilitating the search. The entries in a lexicon could be grouped and given word category (by articles, nouns, pronouns, verbs, adjectives, adverbs and so on) and all words contained within the lexicon listed within the categories to which they belong. The entries are like a, an (determiner), be (verb), boy, stick, glass (noun), green, yellow, red (adjectives), I, we, you, he, she, they (pronouns) etc.

In most contemporary grammatical formalisms, the output of parsing is something logically equivalent to a tree, displaying dominance and precedence relations between constituents of a sentence. Parsing algorithms are usually designed for classes of grammar rather than tailored towards individual grammars.

Types of Parsing

The parsing technique can be categorized into two types such as

1. Top down Parsing
2. Bottom up Parsing

Let us discuss about these two parsing techniques and how they will work for input sentences.

1 Top down Parsing

Top down parsing starts with the starting symbol and proceeds towards the goal. We can say it is the process of construction the parse tree starting at the root and proceeds towards the leaves. It is a strategy of analyzing unknown data relationships by hypothesizing general parse tree structures and then considering whether the known fundamental structures are compatible with the hypothesis. In top down parsing words of the sentence are replaced by their categories like verb phrase (VP), Noun phrase (NP), Preposition phrase (PP), Pronoun (PRO) etc. Let us consider some examples to illustrate top down parsing. We will consider both the symbolical representation and the graphical representation. We will take the words of the sentences and reach at the complete sentence. For parsing we will consider the previous symbols like PP, NP, VP, ART, N, V and so on. Examples of top down parsing are LL (Left-to-right, left most derivation), recursive descent parser etc.

Example 1: Rahul is eating an apple.

Symbolical Representation

$S \rightarrow NP$	VP	
$\square N$	VP	$(\therefore NP \square N)$
$\square N$	AUX	VP
		$(\square VP \square AUX \quad VP)$
$\square N$	AUX	V
		NP
		$(\square VP \square V \quad NP)$
$\square \square$	$.$	$\square UX$
		V
		ART
		N
		$(\square \square P \square ART \quad N)$
$\square \square$	$\square UX$	V
		ART
		$apple$
$\square N$	AUX	V
		an
		$apple$
$\square N$	AUX	$eating$
		an
		$apple$

□□ is eating an apple

□ Rahul is eating an apple.

Graphical Representation

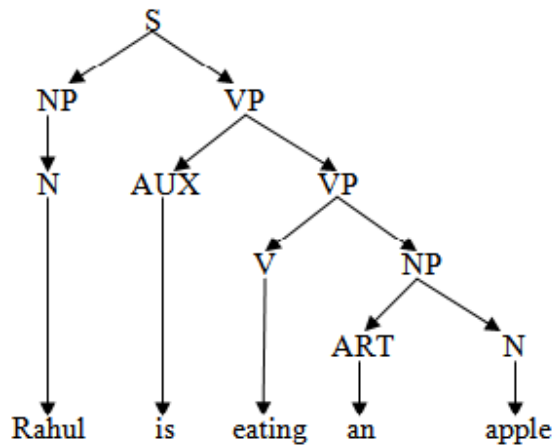


Figure Example of Top down Parsing

Example 2: The small tree shades the new house by the stream.

Symbolical Representation

S → NP VP

- ART NP VP
- The ADJ N VP
- The small N V NP
- The small tree V ART NP
- The small tree shades ART ADJ NP
- The small tree shades the ADJ N NP
- The small tree shades the new N PREP N
- The small tree shades the new house PREP ART N
- The small tree shades the new house by ART N
- The small tree shades the new house by the N

□ The small tree shades the new house by the stream.

Graphical Representation

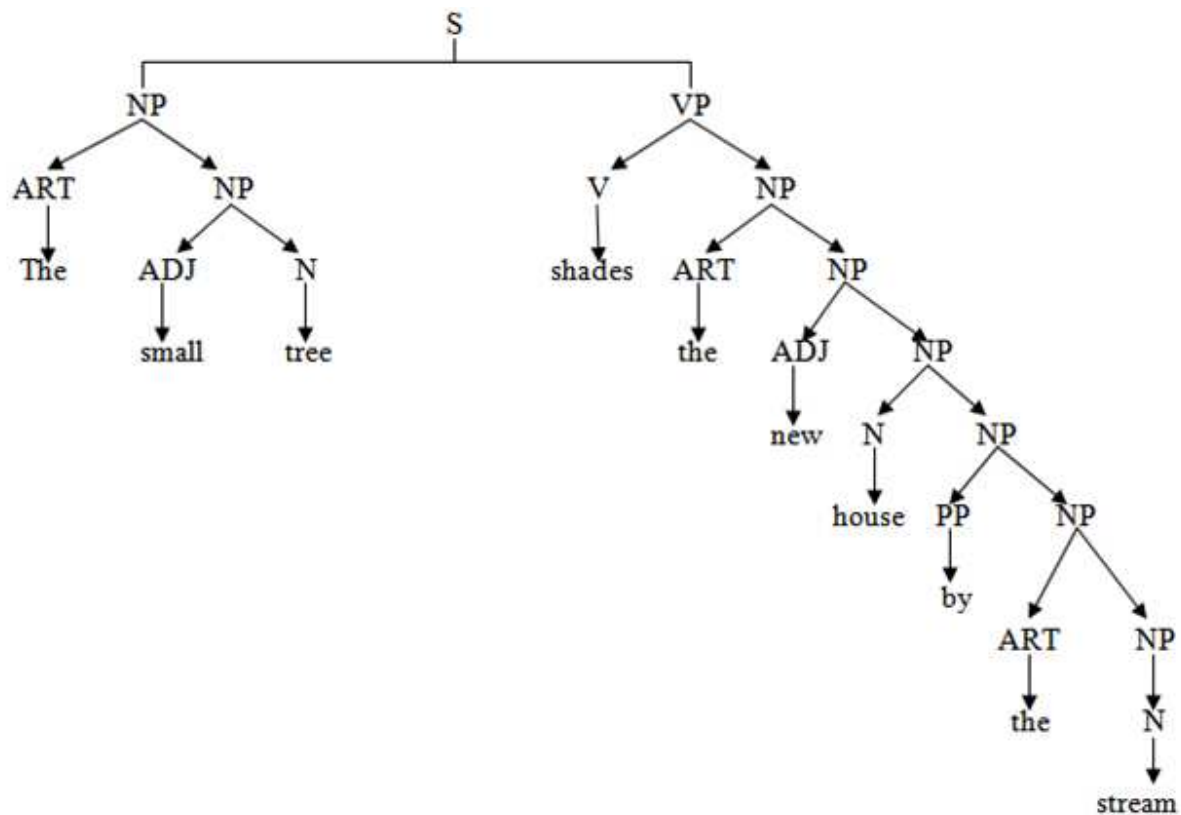


Figure Top down Parsing

Bottom up Parsing

In this parsing technique the process begins with the sentence and the words of the sentence is replaced by their relevant symbols. This process was first suggested by Yngve (1955). It is also called shift reducing parsing. In bottom up parsing the construction of parse tree starts at the leaves and proceeds towards the root. Bottom up parsing is a strategy for analyzing unknown data relationships that attempts to identify the most fundamental units first and then to infer higher order structures for them. This process occurs in the analysis of both natural languages and computer languages. It is common for bottom up parsers to take the form of general parsing engines that can either parse or generate a parser for a specific programming language given a specific of its grammar.

A generalization of this type of algorithm is familiar from computer science LR (k) family can be seen as shift reduce algorithms with a certain amount ("K" words) of look ahead to determine for a set of possible states of the parser which action to take. The sequence of actions from a given grammar can be pre-computed to give a 'parsing table' saying whether a shift or reduce is to be performed and which state to go next. Generally bottom up algorithms are more efficient than top down algorithms, one particular

phenomenon that they deal with only clumsily are “empty rules”: rules in which the right hand side is the empty string. Bottom up parsers find instances of such rules applying at every possible point in the input which can lead to much wasted effort. Let us see some examples to illustrate the bottom up parsing.

Example-1: Rahul is eating an apple.

- □ is eating an apple.
- N AUX eating an apple.
- N AUX V an apple.
- N AUX V ART apple.
- N AUX V ART N
- N AUX V NP
- N AUX VP
- N VP
- NP VP
- S

Graphical Representation

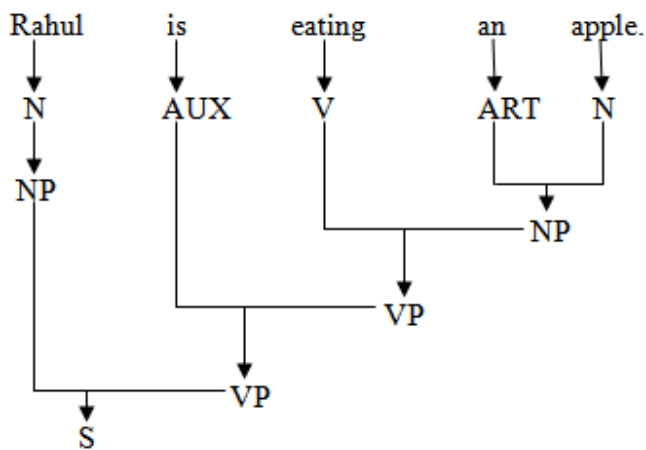


Figure Examples of Bottom up Parsing

Example-2:

- The small tree shades the new house by the stream

- ART small tree shades the new house by the stream
- ART ADJ tree shades the new house by the stream
- ART ADJ N shades the new house by the stream
- ART ADJ N V the new house by the stream
- ART ADJ N V ART new house by the stream
- ART ADJ N V ART ADJ house by the stream
- ART ADJ N V ART ADJ N by the stream
- ART ADJ N V ART ADJ N PREP the stream
- ART ADJ N V ART ADJ N PREP ART stream
- ART ADJ N V ART ADJ N PREP ART N
- ART ADJ N V ART ADJ N PREP NP
- ART ADJ N V ART ADJ N PP
- ART ADJ N V ART ADJ NP
- ART ADJ N V ART NP
- ART ADJ N V NP
- ART ADJ N VP
- ART NP VP
- NP VP
- S

Graphical Representation

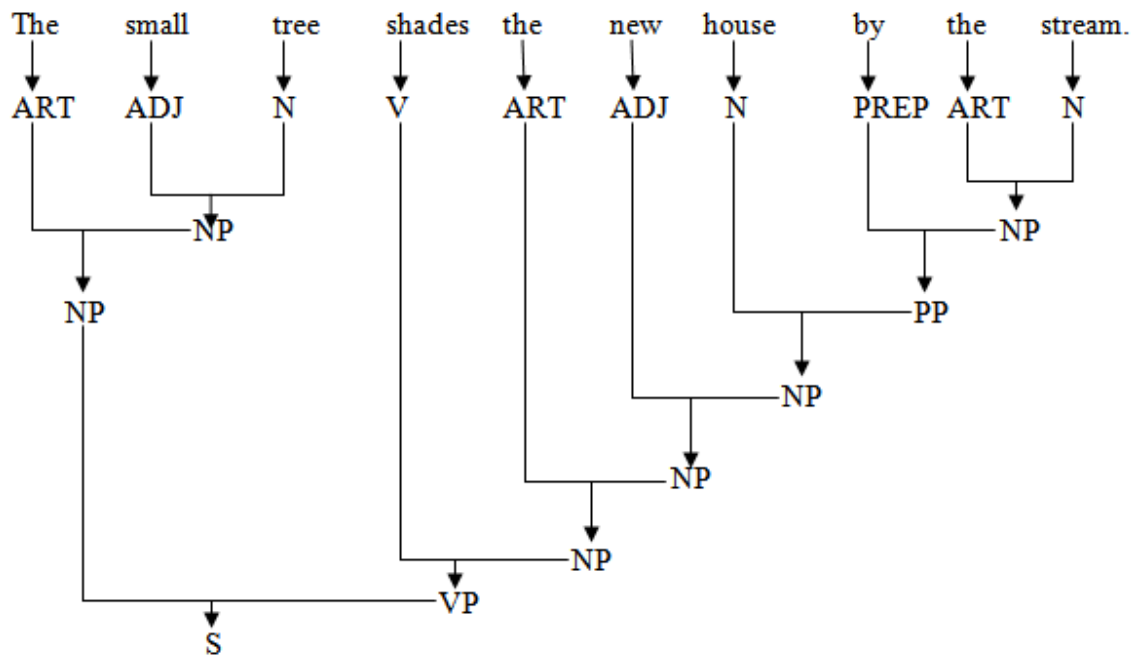


Figure Example of Bottom up Parsing

Deterministic Parsing

A deterministic parser is one which permits only one choice for each word category. That means there is only one replacement possibility for every word category. Thus, each word has a different test conditions. At each stage of parsing always the correct choice is to be taken. In deterministic parsing back tracking to some previous positions is not possible. Always the parser has to move forward. Suppose the parser some form of incorrect choice, then the parser will not proceed forward. This situation arises when one word satisfies more than one word categories, such as noun and verb or adjective and verb. The deterministic parsing network is shown in figure.

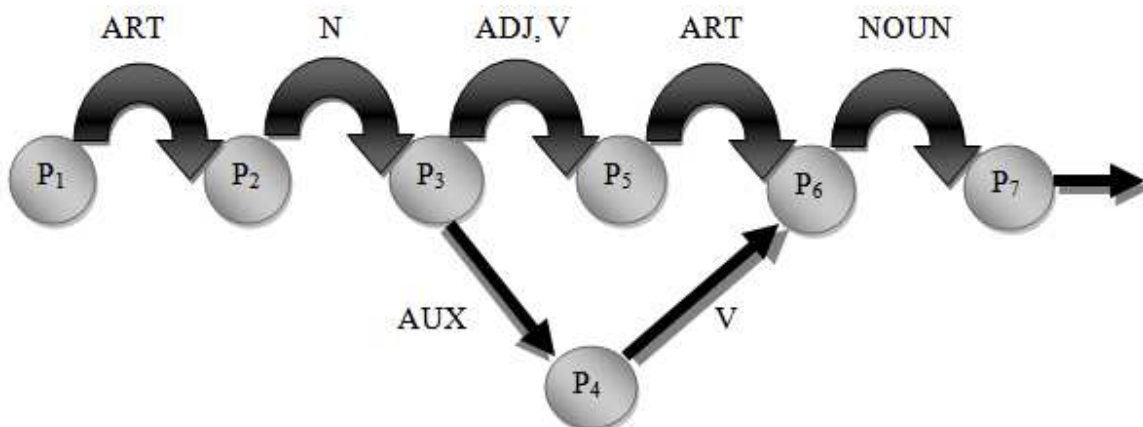


Figure A deterministic Network

Non-Deterministic Parsing

The non deterministic parsing allows different arcs to be labeled with the same test. Thus, they can uniquely make the choice about the next arc to be taken. In non deterministic parsing, the back tracking procedure can be possible. Suppose at some extent of point, the parser does not find the correct word, then at that stage it may backtracks to some of its previous nodes and then start parsing. But the parser has to guess about the proper constituent and then backtrack if the guess is later proven to be wrong. So comparative to deterministic parsing, this procedure may be helpful for a number of sentences as it can backtrack at any point of state. A non deterministic parsing network is shown in figure.

TRANSITION NETWORK

It is a method to represent the natural languages. It is based on applications of directed graphs and finite

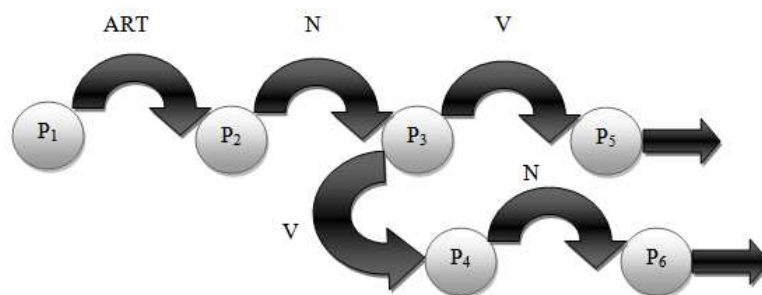


Figure Non-Deterministic Parsing Network

state automata. The transition network can be constructed by the help of some inputs, states and outputs. A transition network may consist of some states or nodes, some labeled arcs from one state to the next state through which it will move. The arc represents the rule or some conditions upon which the transition is made from one state to another state. For example, a transition network is used to recognize a sentence consisting of an article, a noun, an auxiliary, a verb, an article, a noun would be represented by the transition network as follows.

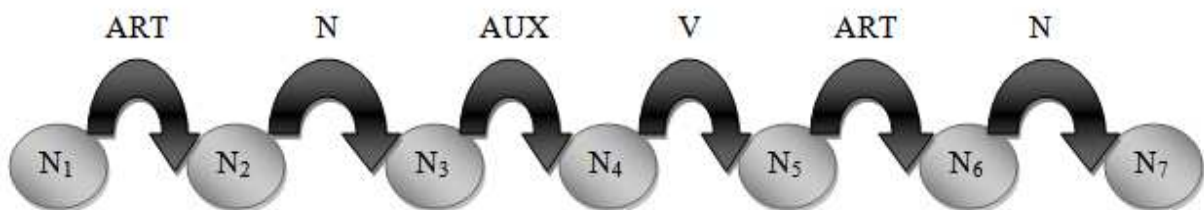


Figure Transition Network

The transition from N_1 to N_2 will be made if an article is the first input symbol. If successful, state N_2 is entered. The transition from N_2 to N_3 can be made if a noun is found next. If successful, state N_3 is entered. The transition from N_3 to N_4 can be made if an auxiliary is found and so on. Suppose consider a sentence "A boy is eating a banana". So if the sentence is parsed in the above transition network then, first 'A' is an article. So successful transition to the node N_1 to N_2 . Then boy is a noun (so N_2 to N_3), "is" is an auxiliary (N_3 to N_4) and finally "banana" is a noun (N_4 to N_5) is done successfully. So the above sentence is successfully parsed in the transition network.

TYPES OF TRANSITION NETWORK

There are generally two types of transition networks like

1. Recursive Transition networks (RTN)
2. Augmented Transition networks (ATN)

Let us focus on these two transition networks and their structure for parsing a sentence.

Recursive Transition Networks (RTN)

RTNs are considered as development for finite state automata with some essential conditions to take the recursive complexion for some definitions in consideration. A recursive transition network consists of nodes (states) and labeled arcs (transitions). It permits arc labels to refer to other networks and they in turn may refer back to the referring network rather than just permitting word categories. It is a modified version of transition network. It allows arc labels that refer to other networks rather than word category. A recursive transition network can have 5 types of arcs (Allen's, JM's) like

- 1) **CAT:** Current word must belong to category.
- 2) **WORD:** Current word must match label exactly.
- 3) **PUSH:** Named network must be successfully traversed.
- 4) **JUMP:** Can always be traversed.
- 5) **POP:** Can always be traversed and indicates that input string has been accepted by the network.

In RTN, one state is specified as a start state. A string is accepted by an RTN if a POP arc is reached and all the input has been consumed. Let us consider a sentence "The stone was dark black".

Here **The:** ART

Stone: ADJ NOUN

Was: VERB

Dark: ADJ

Black: ADJ NOUN

The RTN structure is given in figure

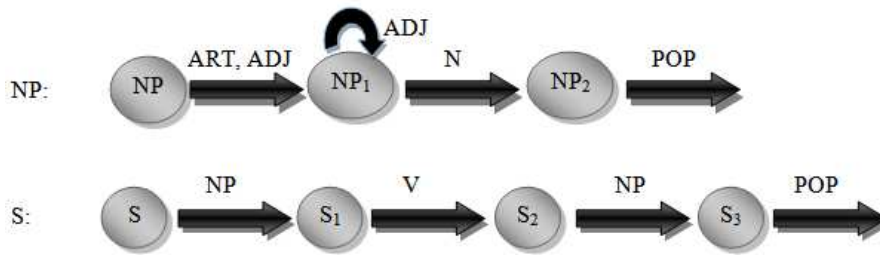


Figure RTN Structure

So we can parse the sentence through the RTN structure as follows.

<u>Current</u>	<u>Word</u>	<u>Return</u>	<u>Arc</u>	<u>Backup States</u>
S	1	-----	NP	-----
NP	1	S ₁	ART	-----
NP ₁	2	S ₁	NOUN	(NP ₁ , 3, S ₁)
NP ₂	3	S ₁	POP	(NP ₁ , 3, S ₁) (NP ₂ , 4, S ₁)
S ₁	3	-----	VERB	(NP ₁ , 3, S ₁) (NP ₂ , 4, S ₁)
S ₂	4	-----	NP	(NP ₁ , 3, S ₁) (NP ₂ , 4, S ₁)
NP	4	S ₃	NONE	(NP ₁ , 3, S ₁) (NP ₂ , 4, S ₁)
NP ₁	3	S ₁	NOUN	(NP ₂ , 4, S ₁)
NP ₂	4	S ₁	POP	(NP ₂ , 4, S ₁)
S ₁	4	-----	VERB	(NP ₂ , 4, S ₁)
S ₂	5	-----	NP	(NP ₂ , 4, S ₁)
NP	5	S ₃	ADJ	(NP ₂ , 4, S ₁)
NP ₁	6	S ₃	NOUN	(NP ₂ , 4, S ₁) (NP ₁ , 7, S ₃)
NP ₂	7	S ₃	POP	(NP ₂ , 4, S ₁) (NP ₁ , 7, S ₃)
NP ₂	7	S ₃	POP	(NP ₂ , 4, S ₁) (NP ₁ , 7, S ₃)
S ₃	7	-----	POP	(NP ₂ , 4, S ₁) (NP ₁ , 7, S ₃)

Finally as there are no words left so the parse is successful.

Also there is an another structure of RTN is described by William Woods (1970) is illustrated in figure. He described the total RTN structure into three parts like sentence (S), Noun Phrase (NP), Preposition Phrase (PP).

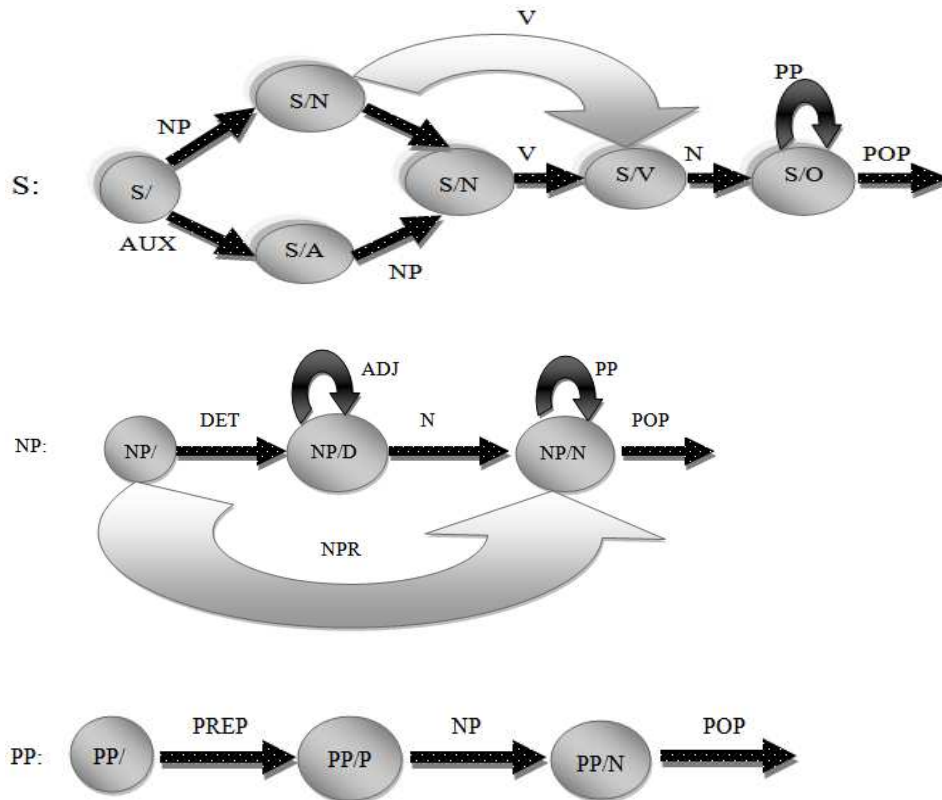


Figure RTN Structure

The number of sentences accepted by an RTN can be extended if backtracking is permitted when a failure occurs. This requires that states having alternative transitions be remembered until the parse progresses past possible failure points. In this way, if a failure occurs at some point, the interpreter can backtrack and try alternative paths. The disadvantage with this approach is that parts of a sentence may be parsed more than time resulting in excessive computations. During the traversal of an RTN, a record must be maintained of the word position in the input sentence and the current state and return nodes to be used as return points when control has been transferred to a lower level network.

Augmented Transition Network (ATN)

An ATN is a modified transition network. It is an extension of RTN. The ATN uses a top down parsing procedure to gather various types of information to be later used for understanding system. It produces the data structure suitable for further processing and capable of storing semantic details. An augmented transition network (ATN) is a recursive transition network that can perform tests and take actions during arc transitions. An ATN uses a set of registers to store information. A set of actions is defined for each arc and the actions can look at and modify the registers. An arc may have a test associated with it. The arc is traversed (and its action) is taken only if the test succeeds. When a lexical arc is traversed, it is put in a special variable (*) that keeps track of the current word. The ATN was first used in LUNAR system. In

ATN, the arc can have a further arbitrary test and an arbitrary action. The structure of ATN is illustrated in figure. Like RTN, the structure of ATN is also consisting of the substructures of S, NP and PP.

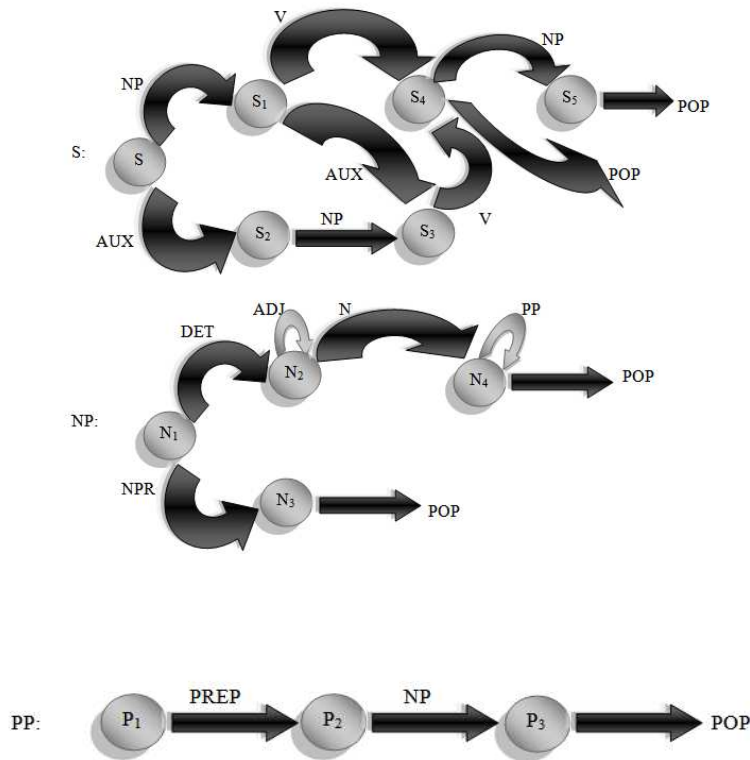


Figure ATN Structure

The ATN collects the sentence features for further analysis. The additional features that can be captured by the ATN are; subject NP, the object NP, the subject verb agreement, the declarative or interrogative mood, tense and so on. So we can conclude that ATN requires some more analysis steps compared to that of RTN. If these extra analysis tests are not performed, then there must some ambiguity in ATN. The ATN represents sentence structure by using a slot filter representation, which reflects more of the functional role of phrases in a sentence. For example, one noun phrase may be identified as “subject” (SUBJ) and another as the “object” of the verb. Within noun phrases, parsing will also identify the determiner structure, adjectives, the noun etc. For the sentence “Ram ate an apple”, we can represent as in figure.

```
(S SUBJ (NP NAME Ram)
      MAIN_V ate
      TENSE PAST
      OBJ (NP DET an
            HEAD apple))
```

Figure Representation of sentence in ATN

The ATN maintains the information by having various registers like DET, ADJ and HEAD etc. Registers are set by actions that can be specified on the arcs. When the arc is followed, the specified action associated with it is executed. An ATN can recognize any language that a general purpose computer can recognize. The ATNs have been used successfully in a number of natural language systems as well as front ends for databases and expert systems.

Module 3

LEARNING

Learning process is the basis of knowledge acquisition process. Knowledge acquisition is the expanding the capabilities of a system or improving its performance at some specified task. So we can say knowledge acquisition is the goal oriented creation and refinement of knowledge. The acquired knowledge may consist of various facts, rules, concepts, procedures, heuristics, formulas, relationships or any other useful information. Knowledge can be acquired from various sources like, domain of interests, text books, technical papers, databases, reports. The terms of increasing levels of abstraction, knowledge includes data, information and Meta knowledge. Meta knowledge includes the ability to evaluate the knowledge available, the additional knowledge required and the systematic implied by the present rules.

Learning involves generalization from experience. Computer system is said to have learning if it is able to not only do the “repetition of same task” more effectively, but also the similar tasks of the related domain. Learning is possible due to some factors like the skill refinement and knowledge acquisition. Skill refinement refers to the situation of improving the skill by performing the same task again and again. If machines are able to improve their skills with the handling of task, they can be said having skill of learning. On the other hand, as we are able to remember the experience or gain some knowledge by

handling the task, so we can improve our skill. We would like our learning algorithms to be efficient in three respects:

- (1) **Computational:** Number of computations during training and during recognition.
- (2) **Statistical:** Number of examples required for good generalization, especially labeled data.
- (3) **Human Involvement:** Specify the prior knowledge built into the model before training.

A similar machine learning architecture is given in figure .



Figure Architecture of Machine Learning

Design of learning element is dictated by the followings.

- (1) What type of performance element is used?
- (2) Which functional component is to be learned?
- (3) How that functional component is represented?
- (4) What kind of feedback is available?
- (5) How can be compared between the existing feedbacks with the new data?
- (6) What are the levels of comparisons? Etc.

Any system designed to create new knowledge and thereby improve its performance must include a set of data structures that represents the system's present level of expertise and a task algorithm that uses the

rules to guide the system's problem solving activity. The architecture of a general learning procedure is given in figure .

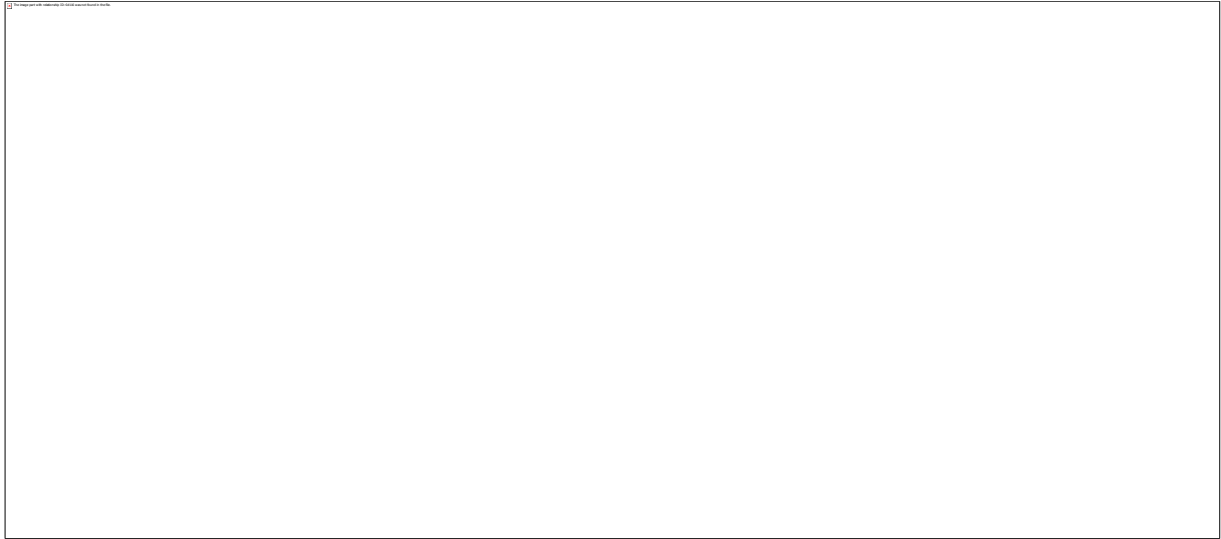


Figure **A general Learning Procedure**

Hence the inputs may be any types of inputs, those are executed for solution of a problem. Those inputs are processed to get the corresponding results. The learning element learns some sort of knowledges by the knowledge acquisition techniques. The acquired knowledge may be required for a same problem in future, for which that problem can be easily solved.

Every learning model must contain implicit or explicit restrictions on the class of functions that can learn. Among the set of all possible functions, we are particularly interested in a subset that contains all the tasks involved in intelligent behaviour. Examples of such tasks include visual perception, auditory perception, planning, control etc. The set does not just include specific visual perception tasks, but the set of all the tasks that an intelligent agent should be able to learn. Although we may like to think that the human brain is some what general purpose, it is extremely restricted in its ability to learn high dimensional functions.

CLASSIFICATION OF LEARNING

The process of learning may be of various types. One can develop learning taxonomies based on the type of knowledge representation used (predicate calculus, rules, frames, scripts etc), the type of knowledge learned (game playing, problem solving) or by areas of application (medical diagnosis, engineering etc). Generally learning may be of two types like single agent learning and multi-agent learning. A general architecture of learning process is given figure .

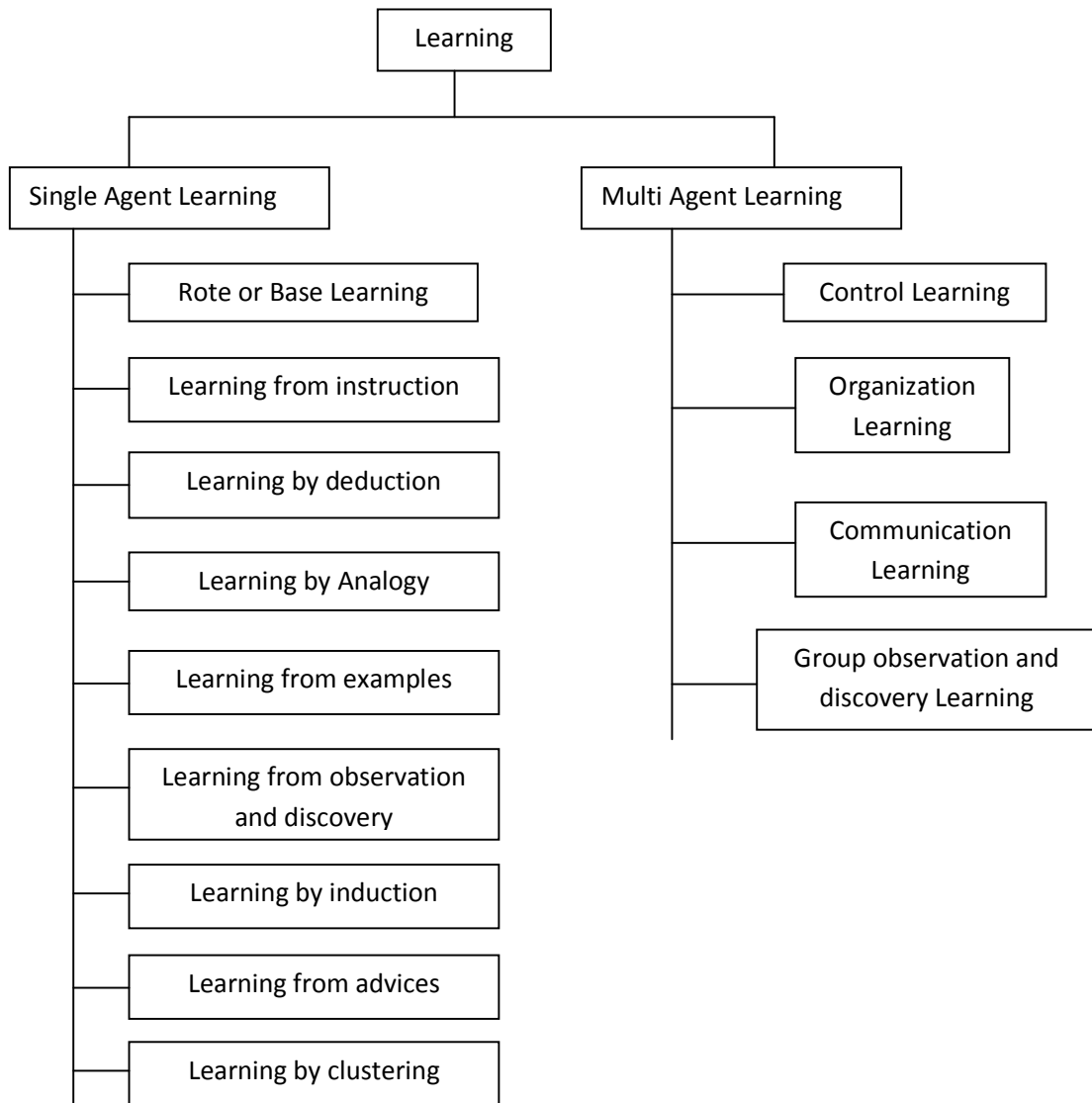


Figure Learning Classification

Single Agent Learning

Over the last four decades, machine learning's primary interest has been single agent learning. Single agent learning involves improving the performance or increasing the knowledge of a single agent. An improvement in performance or an increase in knowledge allows the agent to solve past problems with better quality or efficiency. An increase in knowledge may also allow the agent to solve new problems. An increase in performance is not necessarily due to an increase in knowledge. It may be brought about simply by rearranging the existing knowledge or utilizing it in a different manner. Single agent learning systems may be classified according to their underlying learning strategies. These strategies are classified as follows.

Rote Learning

This strategy does not require the learning system to transform or infer knowledge. It is the simplest form of learning. It requires the least amount of inference and is accomplished by simply copying the knowledge in the same form that it will be used directly into the knowledge base. It includes learning by imitation, simple memorization and learning by being performed. For example we may use this type of learning when we memorize multiplication tables. In this method we store the previous computed values, for which we do not have to recompute them later. Also we can say rote learning is one type of existing or base learning. For example, in our childhood, we have the knowledge that “sun rises in the east”. So in our later stage of learning we can easily memorize the thing. Hence in this context, a system may simply memorize previous solutions and recall them when confronted with the same problem. Generally access of stored value must be faster than it would be to recompute. Methods like hashing, indexing and sorting can be employed to enable this. One drawback of rote learning is it is not very effective in a rapidly changing environment. If the environment does change then we must detect and record exactly what has changed. Also this technique must not decrease the efficiency of the system. We must be able to decide whether it is worth storing the value in the first place.

Learning from Instruction

This strategy also known as learning by being told or learning by direct instruction. It requires the learning system to select and transform knowledge into a usable form and then integrate it into the existing knowledge of the system. It is a more complex form of learning. This learning technique requires more inference than rote learning. It includes learning from teachers and learning by using books, publications and other types of instructions.

Learning by Deduction

This process is accomplished through a sequence of deductive inference steps using known facts. From the known facts, new facts or relationships are logically derived. Using this strategy, the learning system derives new facts from existing information or knowledge by employing deductive inference. It requires more inferences than other techniques. The inference method used is a deductive type, which is a valid form of inference. For example we can say x is the cousin of y if we have the knowledge of x's and y's parents and the rules for cousin relationships. The learner draws deductive inferences from the knowledge and reformulates them in the form of useful conclusions which preserve the information content of the original data. Deductive learning includes knowledge reformulation, compilation and organizational procedures that preserve the truth of the original formulation.

Learning by Analogy

It is a process of learning a new concept or solution through the use of similar known concepts or solutions. We make frequent use of analogical learning. The first step is inductive inference, required to find a common substructure between the problem domain and one of the analogous domains stored in the

learner's existing knowledge base. This form of learning requires the learning system to transform and supplement its existing knowledge from one domain or problem area into new domain. This strategy requires more inferencing by the learning system than previous strategies. Relevant knowledge must be found in the systems existing knowledge by using induction strategies. This knowledge must then be transformed to the new problem using deductive inference. Example of learning by analogy may include the driving technique of vehicles. If we know the driving procedure of a bike, then when we will drive a car then some sort of previous learning procedures we may employ. Similarly for driving a bus or truck, we may use the procedure for driving a car.

Learning from Examples

In this process of learning it includes the learning through various interactive and innovative examples. This strategy, also called concept acquisition. It requires the learning system to induce general class or concept descriptions from examples. Since the learning system does not have prior or analogous knowledge of the concept area, the amount of inferencing is greater than both learning by deduction and analogy. For solving a newly designed problem we may use its corresponding old examples.

Learning from Observations and Discovery

Using this strategy, the learning system must either induce class descriptions from observing the environment or manipulate the environment to acquire class descriptions or concepts. This is an unsupervised learning technique. It requires the greatest amount of inferencing among all of the different forms of learning. From an existing knowledge base, some new forms of discovery of knowledge may formed. The learning discovery process is very important in the respect of constructing new knowledge base.

Learning by Induction

Inductive learning is the system that tries to induce a general rule based on observed instances. In other words, the system tries to infer an association between specific inputs and outputs. In general, the input of the program is a set of training instance where the output is a method of classifying subsequent instance. For example, the input of the program may be color of types of fruits where the output may be the types of fruits those are useful for protein. Induction method involves the learning by examples, experimentation, observation and discovery. The search spaces encountered in learning tend to be extremely large, even by the standards of search based problem solving.

This complexity of problems is cleared by choosing a problem among the different generalizations supported by any given training data. Inductive bias refers to any method a learning program uses to constrain the space of possible generalizations. A no. of inductive learning algorithms have been developed like Probably Approximately Correct (PAC), Version spaces etc. Probably Approximately Correct learning was proposed concerning the situation that cannot be deductive. Approximately correct

is recognized whenever the program can get most of the problems right. In order to increase performance of the program, learning algorithms should restrict the size of hypothesis space. On the other hand, the goal of version space is to produce a description that uses only positive examples. The program in practice, produce a description of all acceptable concepts. In detail, we may conclude that there are two sets of concepts that are produced during learning. Firstly, the most specific concept describes what the target set should be. Secondly, the least specific concept describes what should not be in the target group.

Generally inductive learning is frequently used by humans. This form of learning is more powerful than the others. We use this learning when we formulate a general concept after seeing a number of instances. For example, we can say the taste of sugar is sweet if we have the knowledge about sweetness.

Learning from Advices

In this process we can learn through taking advice from others. The idea of advice taking learning was proposed in early 1958 by McCarthy. In our daily life, this learning process is quite common. Right from our parents, relatives to our teachers, when we start our educational life, we take various advices from others. All most all the initial things and all type of knowledges we acquire through the advices of others. We know the computer programs are written by programmers. When a programmer writes a computer program he or she gives many instructions to computer to follow, the same way a teacher gives his/her advice to his students. The computer follows the instructions given by the programmer. Hence, a kind of learning takes place when computer runs a particular program by taking advice from the creator of the program.

Learning by Clustering

This process is similar to the inductive learning. Clustering is a process of grouping or classifying objects on the basis of a close association or shared characteristics. The clustering process is essentially required in a learning process in which similarity patterns are found among a group of objects. The program must discover for itself the natural classes that exist for the objects, in addition to a method for classifying instances. AUTOCLASS (Cheeseman et al., 1988) is one program that accepts a number of training cases and hypothesizes a set of classes. For any given case, the program provides a set of probabilities that predict into which classes the case is likely to be fall.

Multi Agent Learning

Distributed artificial intelligence (DAI) systems solve problems using multiple, cooperative agents. In these systems, control and information are often distributed among the agents. This reduces the complexity of each agent and allows agents to work in parallel and increases problem solving speed. Also each agent has resource limitations which could limit the ability of a single agent system to solve large, complex problems. Allowing multiple agents to work on these types of problems may be the only way to realistically solve them. In general, multiple agent learning involves improving the performance of the

group of agents as a whole or increasing the domain knowledge of the group. It also includes increasing communication knowledge. An increase in communication knowledge can lead to an increase in performance by allowing the agents to communicate in a more efficient manner. In the context of improving the performance of a group of agents, allowing individual agents to improve their performance may not be enough to improve the performance of the group. To apply learning to the overall group performance, the agents need to adapt and learn to work with each other. The agents may not need to learn more about the domain, as in the traditional sense of machine learning, to improve group performance. In fact to improve the performance of the group, the agents may only need to learn to work together and not necessarily improve their individual performance. In addition, not all the agents must be able to learn or adapt to allow the group to improve.

Control Learning

Learning and adapting to work with other agents involves adjusting the control of each agent's problem solving plan. Different tasks may have to be solved in a specific sequence. If the tasks are assigned to separate agents, the agents must work together to solve the tasks. Learning which agents are typically assigned different types of tasks will allow each agent to select other agents to work with on different tasks. Teams can be formed based on the type of task to be solved. Some of the issues involved are the type, immediacy and importance of task, as well as each agent's task solving ability, capability, reliability and past task assignments. Each team member's plan would be adjusted according to the other agent's plans.

Organization Learning

Learning what type of information and knowledge each agent possesses allows for an increase in performance by specifying the long term responsibilities of each agent. By assigning different agents different responsibilities, the group of agents can improve group performance by providing a global strategy. Organizing the responsibilities reduces the working complexity of each agent.

Communication Learning

Learning what type of information, knowledge reliability and capability each agent possesses allows for an increase in performance by allowing improved communication. Directly addressing the best agent for needed information or knowledge allows for more efficient communication among the agents.

Group Observation and Discovery Learning

Individual agents incorporate different information and knowledge. Combining this differing information and knowledge may assist in the process of learning new class descriptions or concepts that could not have been learned by the agents separately. This type of learning is more effective than the others. The observation towards the procedure will be focused by a group of agents. When a group of different

visions will reach, at that point of view a newly interactive procedure will be found out; which is the discovery of all the agents.

Explanation based Learning

Explanation based learning has ability to learn from a single training instance. Instead of taking more examples the explanation based learning is emphasized to learn a single, specific example. For example, consider the Ludoo game. In a Ludoo game, there are generally four colors of buttons. For a single color there are four different squares. Suppose the colors are red, green, blue and yellow. So maximum four members are possible for this game. Two members are considered for one side (suppose green and red) and other two are considered for another side (suppose blue and yellow). So for any one opponent the other will play his game. A square sized small box marked by symbols one to six is circulated among the four members. The number one is the lowest number and the number six is the highest for which all the operations are done. Always any one from the 1st side will try to attack any one member in the 2nd side and vice versa. At any instance of play the players of one side can attack towards the players of another side. Likewise, all the buttons may be attacked and rejected one by one and finally one side will win the game. Here at a time the players of one side can attack towards the players of another side. So for a specific player, the whole game may be affected. Hence we can say that always explanation based learning is concentrated on the inputs like a simple learning program, the idea about the goal state, the idea about the usable concepts and a set of rules that describes relationships between the objects and the actions.

Explanation based generalization (EBG) is an algorithm for explanation based learning, described in Mitchell et al. (1986). It has two steps first, explain method and secondly, generalize method. During the first step, the domain theory is used to prune away all the unimportant aspects of training examples with respect to the goal concept. The second step is to generalize the explanation as far as possible while still describing the goal concept. Consider the problem of learning the concept bucket. We want to generalize from a single example of a bucket. At first collect the following informations.

1. **Input Examples:**

Owner (object, X) \wedge has part (object, Y) \wedge is(object, Deep) \wedge Color (Object, Green)
 \wedge (Where Y is any thin material)

2. **Domain Knowledge:**

is (a, Deep) \wedge has part (a, b) \wedge is a(b, handle) \rightarrow liftable (a)
has part (a, b) \wedge is a (b, Bottom) \wedge is (b, flat) \rightarrow Stable (a)
has part (a, b) \wedge is a (b, Y) \wedge is (b, Upward – pointing) \rightarrow Open – vessel (a)

3. **Goal:** Bucket

B is a bucket if B is liftable, stable and open-vessel.

4. **Description of Concept:** These are expressed in purely structural forms like Deep, Flat, rounded etc.

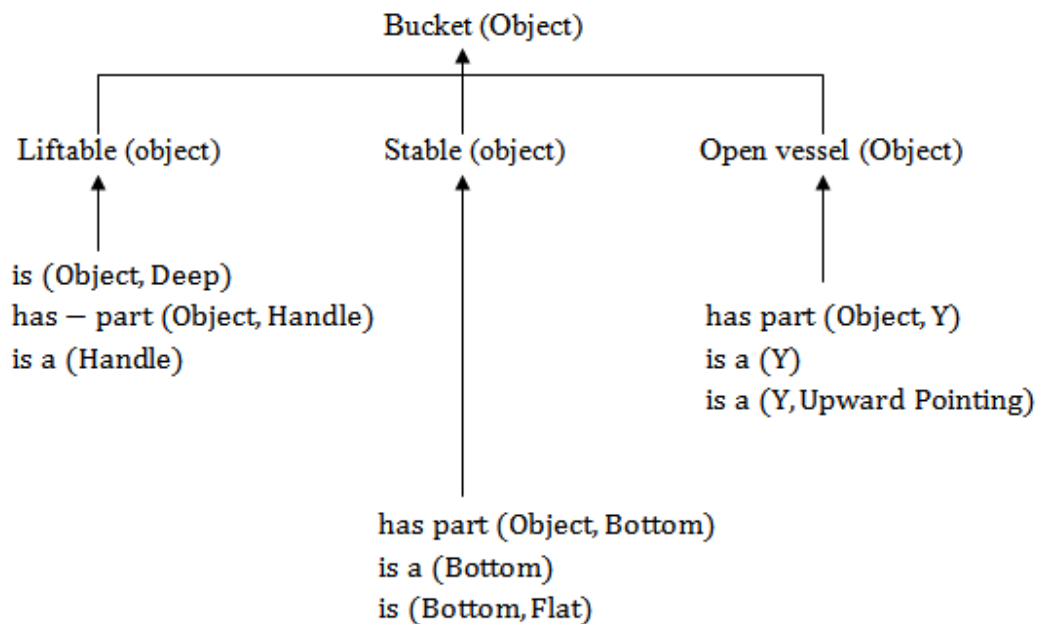


Figure An explanation of BUCKET Object

Given a training example and a functional description, we want to build a general structural description of a bucket. In practice, there are two reasons why the explanation based learning is important.

GENETIC ALGORITHM

Genetic algorithms are based on the theory of natural selection and work on generating a set of random solutions and making them compete in an area where only the fittest survive. Each solution in the set is equivalent to a chromosome. Genetic algorithm learning methods are based on models of natural adaption and evolution. These learning methods improve their performance through processes which model population genetics and survival of the fittest. In the field of genetics, a population is subjected to an environment which places demands on the members. The members which adapt well are selected for mating and reproduction. Generally genetic algorithm uses three basic genetic operators like reproduction, crossover and mutation. These are combined together to evolve a new population. Starting from a random set of solutions the algorithm uses these operators and the fitness function to guide its search for the optimal solution. The fitness function guesses how good the solution in question is and provides a measure to its capability. The genetic operators copy the mechanisms based on the principles of human evolution. The main advantage of the genetic algorithm formulation is that fairly accurate results may be obtained using a very simple algorithm. The genetic algorithm is a method of finding a good answer to a problem, based on the feedback received from its repeated attempts at a solution. The

fitness function is a judge of the GA's attempts for a problem. GA is incapable to derive a problem's solution, but they are capable to know from the fitness function.

Genetic algorithm starts with a fixed size population of data structure which is used to perform some given tasks. After the structure performs the given task or problem, they are rated on their performance by some utility value and a new generation of data structure then created. The new generation is created by mapping with the high performing structure to produce offspring. The offsprings or the children and their parents are retained for the next generation while the poorer performers are not included. Mutations are also performed on the best programming structures to ensure that the full space of possible structure is reachable. This process is repeated for a number of generations until the resultant population consists of only the highest performing structures. Matting between two strings is accomplished with the crossover operation which randomly selects a bit position in the eight bit string and concatenates the head of one parent to the tail of the second parent to produce the off string. Inversion is another type of genetic operation which is applied to a single string.

The GA goes through the following cycle: Generate, Evaluate, Assignment of values, Mate and Mutate. One criteria is to let the GA run for a certain number of cycles. A second one is to allow the GA to run until a reasonable solution is found. Also mutation is a operation, which is used to ensure that all locations of the rule space are reachable, that every potential rule in the rule space is available for evaluation. The mutation operator is typically used only infrequently to prevent random wondering in the search space. Let us focus on the genetic algorithm described as follows.

Step 1:

Generate the initial population.

Step 2:

Calculate the fitness function of each individuals.

Step 3:

Some sort of performance utility values or the fitness values are assigned to individuals.

Step 4:

New populations are generated from the best individuals by the process of selection.

Step 5:

Perform the crossover and mutation operation.

Step 6:

Replace the old population with the new individuals.

Step 7:

Perform step-2 until the goal is reached.

In its simplest form, the standard genetic algorithm is a method of stochastic optimization for discrete programming problems of the form.

$$\begin{array}{c} \text{Maximize } f(P) \\ \text{subject to } P \in \Omega \subseteq \{0, 1\}^n \end{array}$$

In this case $f: \Omega \rightarrow \mathbb{R}$ is called the fitness function and the n -dimensional binary vectors in Ω are called strings. The most noticeable difference between the standard genetic algorithm and the methods of optimization is that at each stage of the computation, genetic algorithms maintain a collection of samples from the search space Ω rather than a single point. This collection of samples is called a population of strings. To start the genetic search, an initial population of say, B binary strings $P(0) = \{P_1, P_2 \dots P_B\} \subset \Omega$; each with n bits, is created. Usually, this initial population is created randomly because it is not known a priori where the globally optimal strings in Ω are likely to be found. If such information is given though it may be used to bias the initial population towards the most promising regions of Ω . From this initial population, subsequent population $P(1), P(2) \dots \dots \dots P(m) \dots \dots \dots$ will be computed by employing the three genetic operators of selection, crossover and mutation.

Applications and Advantages of Genetic Algorithm

Some of the applications and characteristics of genetic algorithm as described as following.

- GA is a randomized search and optimization technique guided by the principle of natural genetic systems.
- The GA is being applied to a wide range of optimization and learning problems in many domains.
- GAs also lend themselves well to power system optimization problems.
- GAs solve problems using principles inspired by natural population genetics.
- GAs can provide globally optimal solutions.
- GAs work with a coding of the parameter set, not the parameters themselves. Therefore, they can easily handle integral variables.
- GAs use probabilistic transition rules, non deterministic rules.
- GAs are used to solve the problems like job shop problem, optimization problems, skill based employee allocation problems, scheduling of job shop problems etc.

→ GA performs always multidirectional search by maintaining a population of potential solutions.

NEURAL NETWORK

A neural network consists of inter connected processing elements called neurons that work together to produce an output function. The output of a neural network relies on the cooperation of the individual neurons within the network to operate. Well designed neural networks are trainable systems that can often “learn” to solve complex problems from a set of exemplars and generalize the “acquired knowledge” to solve unforeseen problems, i.e. they are self-adaptive systems. A neural network is used to refer to a network of biological neurons. A neural network consists of a set of highly interconnected entities called nodes or units. Each unit accepts a weighted set of inputs and responds with an output.

Mathematically let $I = (I_1, I_2, \dots, I_n)$ represent the set of inputs presented to the unit U . Each input has an associated weight that represents the strength of that particular connection. Let $W = (W_1, W_2, \dots, W_n)$ represent the weight vector corresponding to the input vector X . By applying to V , these weighted inputs produce a net sum at U given by

$$S = \text{SUM} (W_i * I_i)$$

A neural network is first and foremost a graph, with patterns represented in terms of numerical values attached to the nodes of the graph and transformations between patterns achieved via simple message-passing algorithms. The graph contains a number of units and weighted unidirectional connections between them. The output of one unit typically becomes an input for another. There may also be units with external inputs and outputs. The nodes in the graph are generally distinguished as being input nodes or output nodes and the graph as a whole can be viewed as a representation of a multivariate functions linking inputs to outputs. Numerical values (weights) are attached to the links of the graphs, parameterizing the input/ output function and allowing it to be adjusted via a learning algorithm. A broader view of a neural network architecture involves treating the network as a statistical processor characterized by making particular probabilistic assumptions about data. Figure illustrates one example of a possible neural network structure.

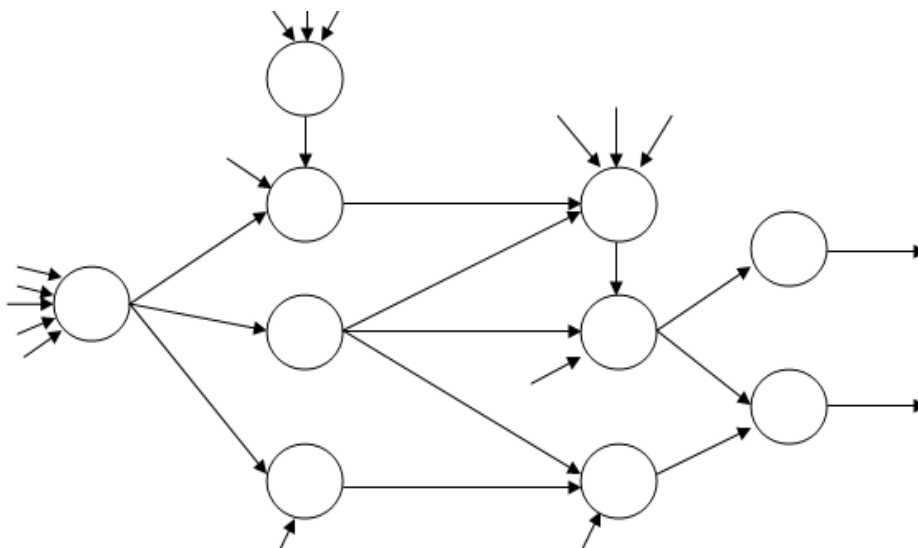


Figure An example of a Neural network structure

Patterns appearing on the input nodes or the output nodes of a network are viewed as samples from probability densities and a network is viewed as a probabilistic model that assigns probabilities to patterns. Biologically, we can also define a neuron. The human body is made up of a vast array of living cells. Certain cells are interconnected in a way that allows them to communicate pain or to actuate fibres or tissues. Some cells control the opening and closing of minuscule valves in the veins and arteries. These specialized communication cells are called neurons. Neurons are equipped with long tentacle like structures that stretch out from the cell body, permitting them to communicate with other neurons. The tentacles that take in signals from other cells and the environment itself are called dendrites, while the tentacles that carry signals from the neuron to other cells are called axons.

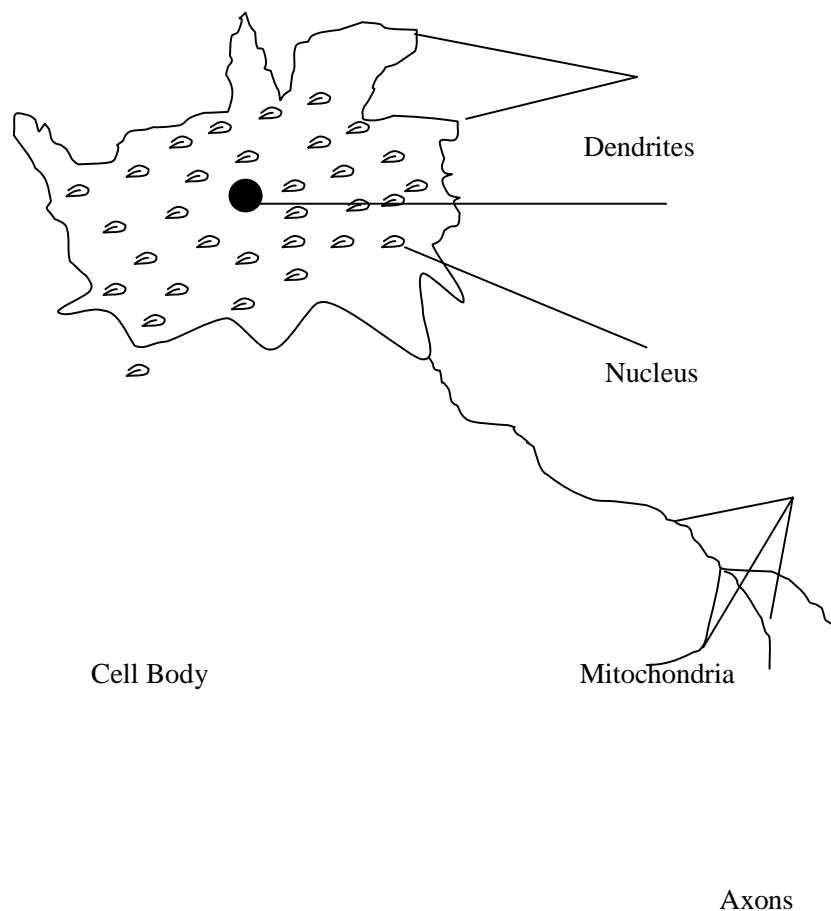


Figure A Neuron

FEATURES OF ARTIFICIAL NETWORK (ANN)

Artificial neural networks may be physical devices or simulated on conventional computers. From a practical point of view, an ANN is just a parallel computational system consisting of many simple processing elements connected together in a specific way in order to perform a particular task. There are some important features of artificial networks as follows.

- (1) Artificial neural networks are extremely powerful computational devices (Universal computers).
- (2) ANNs are modeled on the basis of current brain theories, in which information is represented by weights.
- (3) ANNs have massive parallelism which makes them very efficient.
- (4) They can learn and generalize from training data so there is no need for enormous feats of programming.
- (5) Storage is fault tolerant i.e. some portions of the neural net can be removed and there will be only a small degradation in the quality of stored data.
- (6) They are particularly fault tolerant which is equivalent to the “graceful degradation” found in biological systems.
- (7) Data are naturally stored in the form of associative memory which contrasts with conventional memory, in which data are recalled by specifying address of that data.
- (8) They are very noise tolerant, so they can cope with situations where normal symbolic systems would have difficulty.
- (9) In practice, they can do anything a symbolic/ logic system can do and more.
- (10) Neural networks can extrapolate and intrapolate from their stored information. The neural networks can also be trained. Special training teaches the net to look for significant features or relationships of data.

TYPES OF NEURAL NETWORKS

There are a number of models of neural networks have been developed, each defined at a different level of abstraction and trying to model different aspects of neural systems. They range from models of the short term behaviour of individual neurons, through models of how the dynamics of neural circuitry arise from interactions between individual neurons, to models of how behaviour can arise from abstract neural modules that represent complete sub systems. Some neural networks have been illustrated below.

Single Layer Network

A single layer neural network consists of a set of units organized in a layer. Each unit U_n receives a weighted input I_j with weight W_{jn} . Figure shows a single layer neural network with j inputs and n outputs.

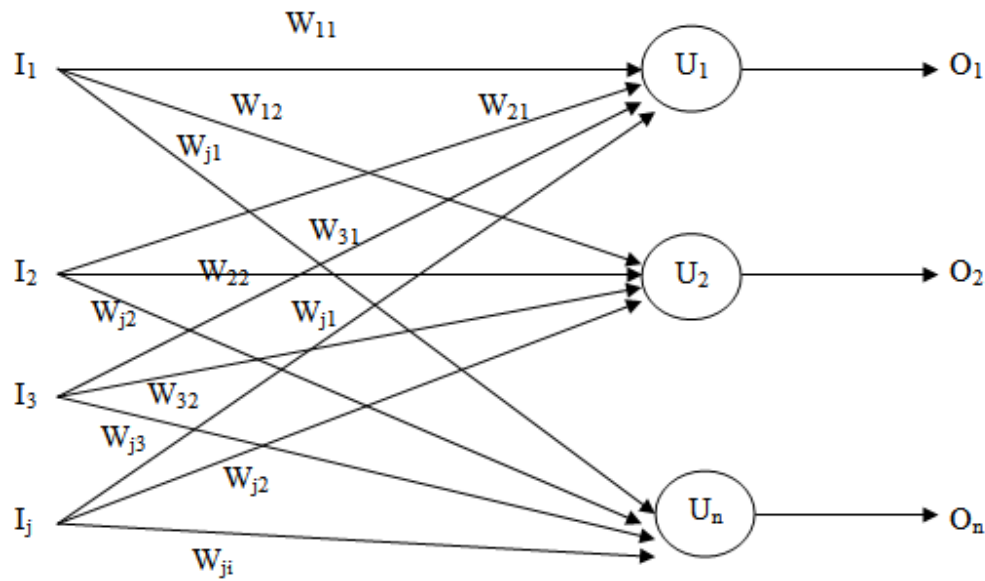


Figure A single Layer neural Network

Let $I = (i_1, i_2 \dots \dots i_j)$ be the input vector and let the activation function f be simply, so that the activation value is just the net sum to a unit. The $j \times n$ weight matrix is calculated as follows.

$$W = \begin{bmatrix} W_{11} & W_{12} & \dots & \dots & W_{1n} \\ W_{21} & W_{22} & \dots & \dots & W_{2n} \\ \vdots & \vdots & & & \vdots \\ W_{j1} & W_{j2} & \dots & \dots & W_{jn} \end{bmatrix}$$

Thus the output O_x at unit U_x is

$$O_k = (W_{1k}, W_{2k} \dots \dots W_{jk}) \begin{pmatrix} I_1 \\ I_2 \\ \vdots \\ I_j \end{pmatrix}$$

Multilayer Network

A multilayer network has two or more layers of units, with the output from one layer serving as input to the next. Generally in a multilayer network there are 3 layers present like, input layer, output layer and hidden layer. The layer with no external output connections are referred to as hidden layers. A multilayer neural network structure is given in figure.

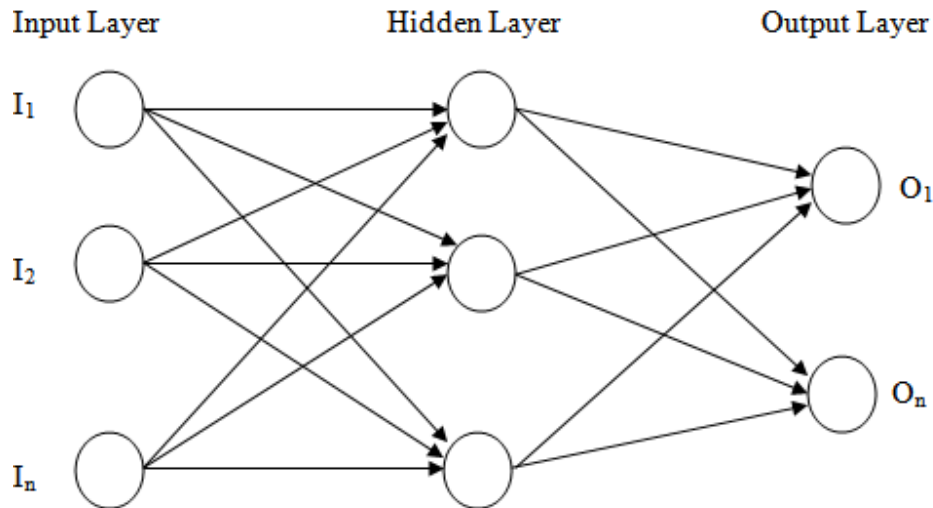


Figure A multilayer neural network

Any multilayer system with fixed weights that has a linear activation function is equivalent to a single layer linear system, for example, the case of a two layer system. The input vector to the first layer is I , the output $O = W_1 * I$ and the second layer produces output $O_2 = W_2 * O$. Hence $O_2 = W_2 * (W_1 * I)$

$$= (W_2 * W_1) * I$$

So a linear system with any number n of layers is equivalent to a single layer linear system whose weight matrix is the product of the n intermediate weight matrices. A multilayer system that is not linear can provide more computational capability than a single layer system. Generally multilayer networks have proven to be very powerful than single layer neural network. Any type of Boolean function can be implemented by such a network. At the output layer of a multilayer neural network the output vector is compared to the expected output. If the difference is zero, no changes are made to the weights of connections. If the difference is not zero, the error is calculated and is propagated back through the network.

Feed Forward neural network

The neural networks consist of multiple layers of computational units, usually interconnected in a feed forward way. The feed forward neural networks are the first, simplest type of artificial neural networks devised. In this network, the information moves in only one direction, forward from the input nodes, through the hidden nodes and to the output nodes. There are no cycles or loops in the network. In other way we can say the feed forward neural network is one that does not have any connections from output to input. All inputs with variable weights are connected with every other node. A single layer feed forward network has one layer of nodes, whereas a multilayer feed forward network has multiple layers of nodes. The structure of a feed forward multilayer network is given in figure.

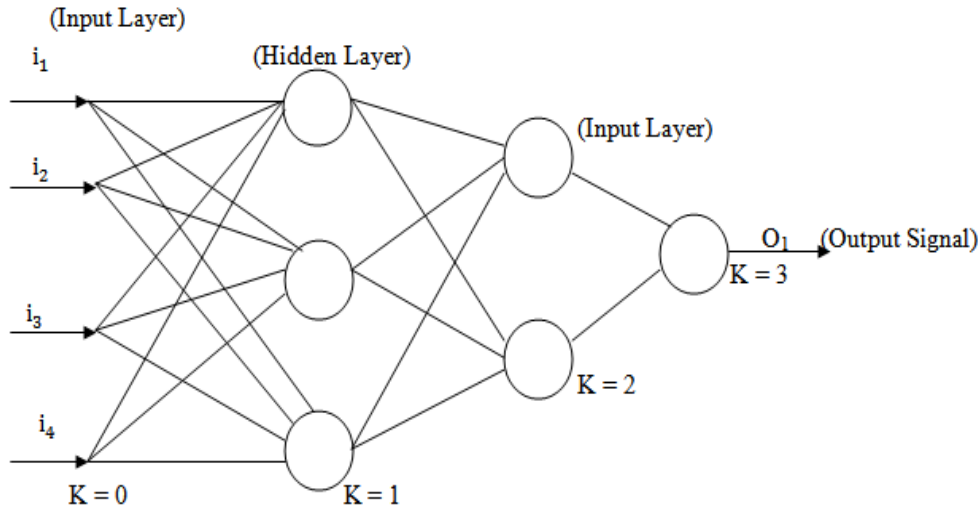


Figure Multilayer Feed Forward Neural Network

Data are introduced into the system through an input layer. This is followed by processing in one or more intermediate (hidden layers). Output data emerge from the network's final layer. The transfer functions contained in the individual neurons can be almost anything. The input layer is also called as Zeroth layer, of the network serves to redistribute input values and does no processing. The output of this layer is described mathematically as follows.

$$O_M^0 = i_m \text{ Where } m = 1, 2 \dots N^0$$

(N^0 represents the no. of neurons in the input or zeroth layer).

The input to each neuron in the first hidden layer in the network is a summation all weighted connections between the input or Zeroth layer and the neuron in the first hidden layer. We will write the weighted sum as net sum or net input. We can write the net input to a neuron from the first layer as the product of that input vector i_m and weight factor w_m plus a bias term θ . The total weighted input to the neuron is a summation of these individual input signals described as follows.

$$\text{net sum} = \sum_{m=1}^N W_m X_m + \theta$$

Where N represents the number of neurons in the input layer.

The net sum to the neuron is transformed by the neuron's activation or transfer function, f to produce a new output value for the neuron. With back propagation, this transfer function is most commonly either a sigmoid or a linear function. In addition to the net sum, a bias term θ is generally added to offset the input. The bias is designed as a weight coming from a unitary valued input and denoted as W_0 . So, the final output of the neuron is given by the following equation.

$$\begin{aligned} \text{Output} &= f(\text{net sum}) \\ &= f\left(\sum_{m=1}^N w_m i_m + \theta\right) \\ &= f\left(\sum_{m=1}^{N^0} w_m i_m^0 + w_0\right) \end{aligned}$$

But one question may arise in reader's mind. Why we are using the hidden layer between the input and output layer? The answer to this question is very silly. Each layer in a multilayer neural network has its own specific function. The input layer accepts input signals from the outside world and redistributes these signals to all neurons in the hidden layer. Actually, the input layer rarely includes computing neurons and thus does not process input patterns. The output layer accepts output signals, or in other words a stimulus patterns, from the hidden layer and established the output patterns of the entire network. Neurons in the hidden layer detect the features, the weights of the neurons represent the features hidden in the input patterns. These features are then used by the output layer in determining the output patterns. With one hidden layer we can represent any continuous function of the input signals and with two hidden layers even discontinuous functions can be represented. A hidden layer hides its desired output. Neurons in the hidden layer cannot be observed through the input/ output behaviour of the network. The desired output of the hidden layer is determined by the layer itself. Generally, we can say there is no obvious way to know what the desired output of the hidden layer should be.

Back Propagation neural network

Multilayer neural networks use a most common technique from a variety of learning technique, called the back propagation algorithm. In back propagation neural network, the output values are compared with the correct answer to compute the value of some predefined error function. By various techniques the error is then fed back through the network. Using this information, the algorithms adjust the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles the network will usually converge to some state where the error of the calculation is small.

The goal of back propagation, as with most training algorithms, is to iteratively adjust the weights in the network to produce the desired output by minimizing the output error. The algorithm's goal is to solve credit assignment problem. Back propagation is a gradient-descent approach in that it uses the minimization of first-order derivatives to find an optimal solution. The standard back propagation algorithm is given below.

Step1:

Build a network with the chosen number of input, hidden and output units.

Step2:

Initialize all the weights to low random values.

Step3:

Randomly, choose a single training pair.

Step4:

Copy the input pattern to the input layer.

Step5:

Cycle the network so that the activation from the inputs generates the activations in the hidden and output layers.

Step6:

Calculate the error derivative between the output activation and the final output.

Step7:

Apply the method of back propagation to the summed products of the weights and errors in the output layer in order to calculate the error in the hidden units.

Step8:

Update the weights attached to each unit according to the error in that unit, the output from the unit below it and the learning parameters, until the error is sufficiently low.

To derive the back propagation algorithm, let us consider the three layer network shown in figure .

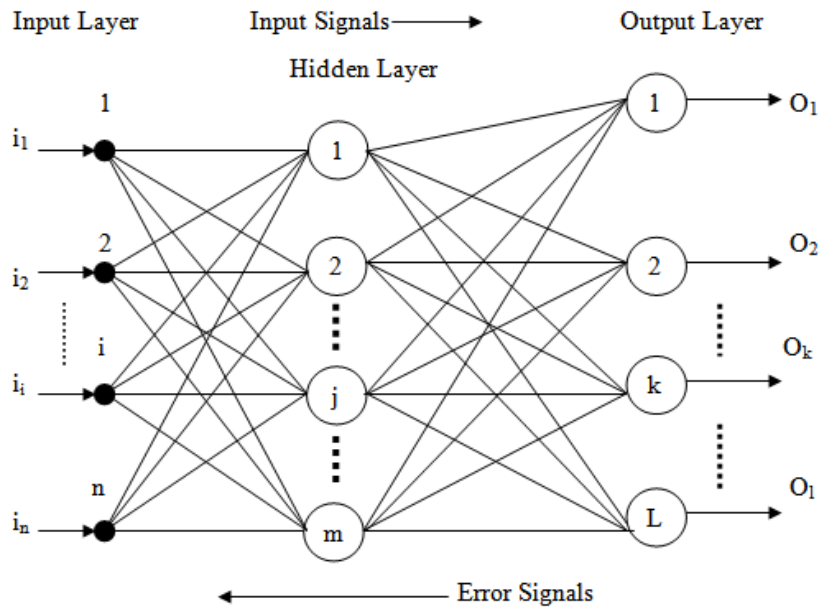


Figure Three layer back-propagation neural network

To propagate error signals, we start at the output layer and work backward to the hidden layer. The error signal at the output of neuron k at iteration x is defined as

$$e_k^x = O_k^x - d_k$$

Where d_k is the desired output of neuron k at iteration x .

Generally, computational learning theory is concerned with training classifiers on a limited amount of data. In the context of neural networks a simple heuristic, called early stopping often ensures that the network will generalize well to examples not in the training set. There are some problems with the back propagation algorithm like speed of convergence and the possibility of ending up in a local minimum of the error function. Today there are a variety of practical solutions that make back propagation in multilayer perceptrons the solution of choice for many machine learning tasks.

CLUSTERING

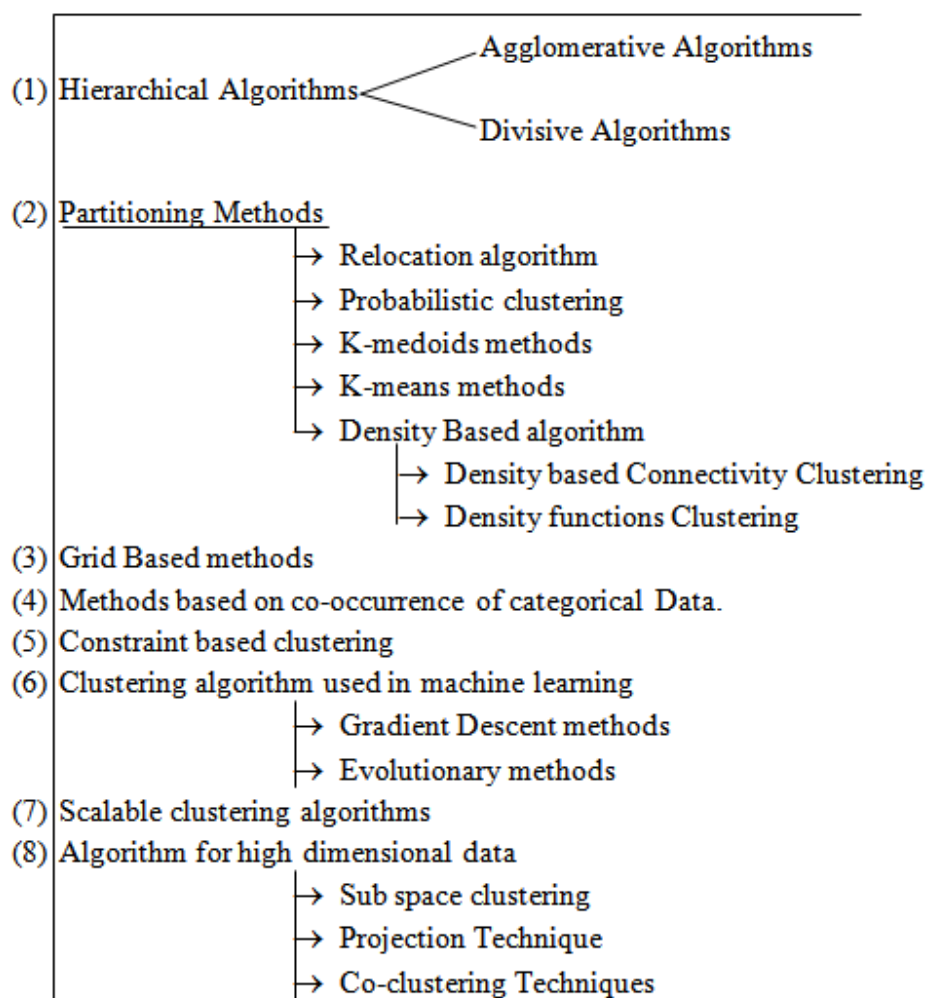
Clustering is a division of data into groups of similar objects. Representing the data by fewer clusters necessarily loses certain fine details, but achieves simplification. It models data by its clusters. Data modeling puts clustering in a historical perspective rooted in mathematics, statistics and numerical analysis. From a machine learning perspective clusters correspond to hidden patterns, the search for clusters in unsupervised learning and the resulting system represents a data concept. From a practical perspective, clustering plays an outstanding role in data mining applications such as scientific data exploration, information retrieval and text mining, spatial database applications, web analysis, marketing, medical diagnostics, computational biology and many others.

Clustering is the subject of active research in several fields such as statistics, pattern recognition and machine learning. Clustering is the classification of similar objects into different group. We can also define clustering is the unsupervised learning of a hidden data concept. Besides the term data clustering, there are a number of terms with similar meanings, including cluster analysis, automatic classification, numerical taxonomy, and typological analysis.

Types of Clustering

Categorization of clustering algorithms is neither straight forward nor canonical. Data clustering algorithms can be hierarchical or partitional. Two-way clustering, co-clustering or bi-clustering are the names for clustering where not only the objects are clustered but also the features of the objects. We provide a classification of clustering algorithms listed below.

Clustering Algorithms



Different Clustering Algorithms

PATTERN RECOGNITION

Pattern recognition as a field of study developed significantly in the 1960's. It was very much interdisciplinary subject, covering developments in the areas of statistics, engineering, artificial intelligence, computer science, psychology and physiology, among others. Some people entered the field with a real problem to solve. The large numbers of applications ranging from the classical ones such as automatic character recognition and medical diagnosis to the more recent ones in data mining (such as credit scoring, consumer sales analysis and credit card transaction analysis), have attracted considerable research effort, with many methods developed and advances made. Pattern recognition of the objects into a lot of categories or classes. It is an integral part in most machine intelligence system built for decision making. The nature of the pattern recognition is engineering. But the final aim of pattern recognition is to design machines to solve the gap between application and theory. It is a process of identifying a stimulus. This process is often accomplished with incomplete or ambiguous information. The basic model of pattern classifiers is shown in figure.

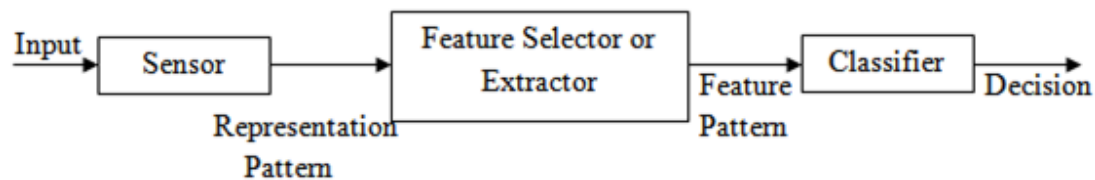


Figure A common Pattern Classifier

A complete pattern recognition system consists of:

- (a) A sensor: It gathers the information to be classified.
- (b) A feature selector or Extractor: Feature selection is the process of selecting a subset of a given set of variables. The feature extractor mechanism takes a possible non linear combination of the original variables to form new variables.
- (c) A classifier: It classifies or describes the observations relying on the extracted features.

To understand the problem of designing a pattern recognition system, we must understand the problems that each of these components must solve. Different components of the pattern recognition system are sensing, segmentation, feature extraction, classification, post processing. The input to a pattern recognition system is some kind of a transducer, such as camera or a microphone array. Sensing is used to eliminate the noise. A sensor converts images or sounds or other physical into signal data. The segmentor isolates sensed objects from the back ground or from other objects. A feature extractor measures object properties that are useful for classification. The goal of feature extractor is to characterize an object to be recognized by measurements whose values are very similar for objects in the same category and for objects in different categories very different. The task of feature extractor is domain dependent and requires the knowledge of the domain. The task of the classifier component proper of a full system is to

use the feature vector provided by the feature extractor to assign the object to a category. The classifier uses the features to assign the sensed object to a category. The post processor uses the output of the classifier to decide on the recommended action.

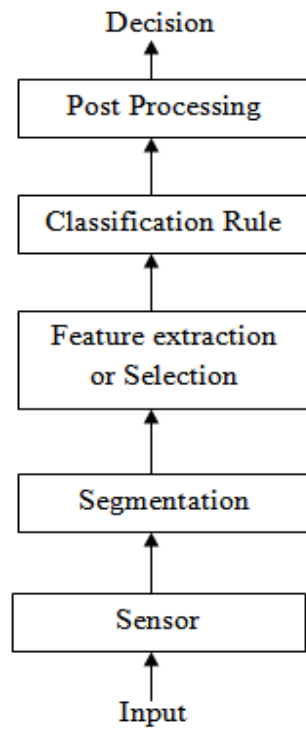


Figure Components of a Pattern Recognition System

The design cycle of a pattern recognition system entails the repetition of a number of different activities training and evaluation. A structure of the design cycle of a pattern recognition system is illustrated in figure .

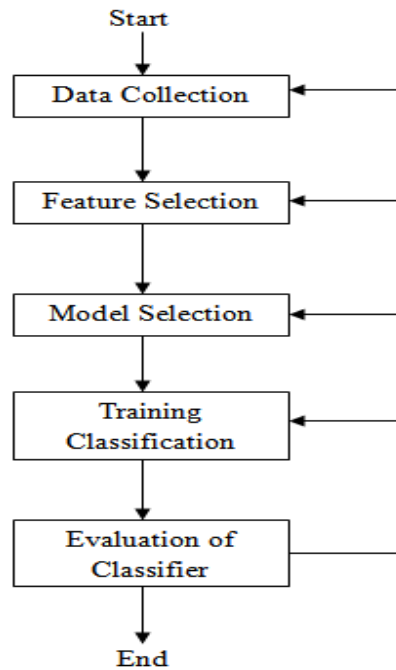


Figure Design cycle of a Pattern Recognition System

Data collection can account for large part of the cost of developing a pattern recognition system. The selection of some separate distinguishable features depends on the characteristics of the problem domain. In selecting features, we would like to find features that are simple to extract, invariant to irrelevant transformation and useful to discriminating patterns in different categories. The selections of models are done by the different descriptions, which are in mathematical form. The training process uses some or all of the data to determine the system parameters. Different types of training protocols are stochastic, batch and online. Evaluation is needed to measure the performance of the system and to identify the need for improvements in its components. There are two main divisions of classification in pattern recognition like supervised classification and unsupervised classification. In supervised classification, we have a set of data samples (each consisting of measurements on a set of variables) with associated labels, the class types. These are used as exemplars in the classifier design. In unsupervised classification, the data are not labeled and we seek to find groups in the data and the features that distinguish one group from another.

EXPERT SYSTEM:

An expert system may be viewed as a computer simulation of a human expert. It can also be defined as a computer program that simulates the judgment and behaviour of a human or an organization that has expert knowledge and experience in a particular field. Typically such a system contains a knowledge base containing accumulated experience and a set of rules for applying the knowledge base to each particular situation that is described to the program. Expert

systems also use human knowledge to solve problems that normally would require human intelligence. These expert systems represent the expertise knowledge as data or rules within the computer. These rules and data can be called upon when needed to solve problems. Books and manual guides have a tremendous amount of knowledge but a human has to read and interpret the knowledge for it to be used.

A system that uses human knowledge captured in a computer to solve problems that ordinarily require human expertise. A computer program designed to model the problem solving ability of a human expert. Expert systems make extensive use of specialized knowledge to solve problems at the level of a human expert. An expert is a person who has expertise in a certain area i.e. the expert has knowledge or special skills that are not known or available to most people. An expert can solve problems that most people cannot solve them much more efficiently. Thus expert system technology may include special expert system languages, programs and hardware designed to aid in the development and execution of expert systems. The knowledge in expert systems may be either expertise or knowledge that is generally available from books, magazines and knowledgeable persons.

DIFFERENCE BETWEEN EXPERT SYSTEM AND CONVENTIONAL SYSTEM

The principle distinction between expert systems and traditional problem solving programs is the way in which the problem related expertise is coded. In conventional applications, problem expertise is encoded in both program and data structures. In the expert system approach all of the problem related expertise is encoded in data structures only, none is in programs. Generally in expert systems, the use of knowledge is vital. But in conventional system data is used more efficiently than knowledge. Conventional systems are not capable of explaining a particular conclusion for a problem. These systems try to solve in a straight forward manner. But expert systems are capable of explaining how a particular conclusion is reached and why requested information is needed during a process. However, the problems are solved more efficiently than a conventional system by an expert system. Generally in an expert system, it uses the symbolic representations for knowledge i.e. the rules, different forms of networks, frames, scripts etc. and performs their inference through symbolic computations. But conventional systems are unable to express these terms. They just simplify the problems in a straight forward manner and are incapable to express the “how, why” questions. Also the problem solving tools those are present in expert system are purely absent in conventional systems. The various types of problems are always solved by the experts in an expert system. So the solution of the problem is more accurate than a conventional system.

THE DEVELOPMENT PROCESS OF AN EXPERT SYSTEM

By the definition, an expert system is a computer program that simulates the thought process of a human expert to solve complex decision problems in a specific domain. The expert system's knowledge is obtained from expert sources which are coded into most suitable form. The process of building an expert system is called knowledge engineering and is done by a knowledge engineer. The knowledge engineer is a human with a background in computer science and AI and he knows how to build expert systems. A knowledge engineer also decides how to represent the knowledge in an expert system and helps the programmers to write the code. Knowledge engineering is the acquisition of knowledge from a human expert or any other source. The different stages in the development of an expert system are illustrated in figure.

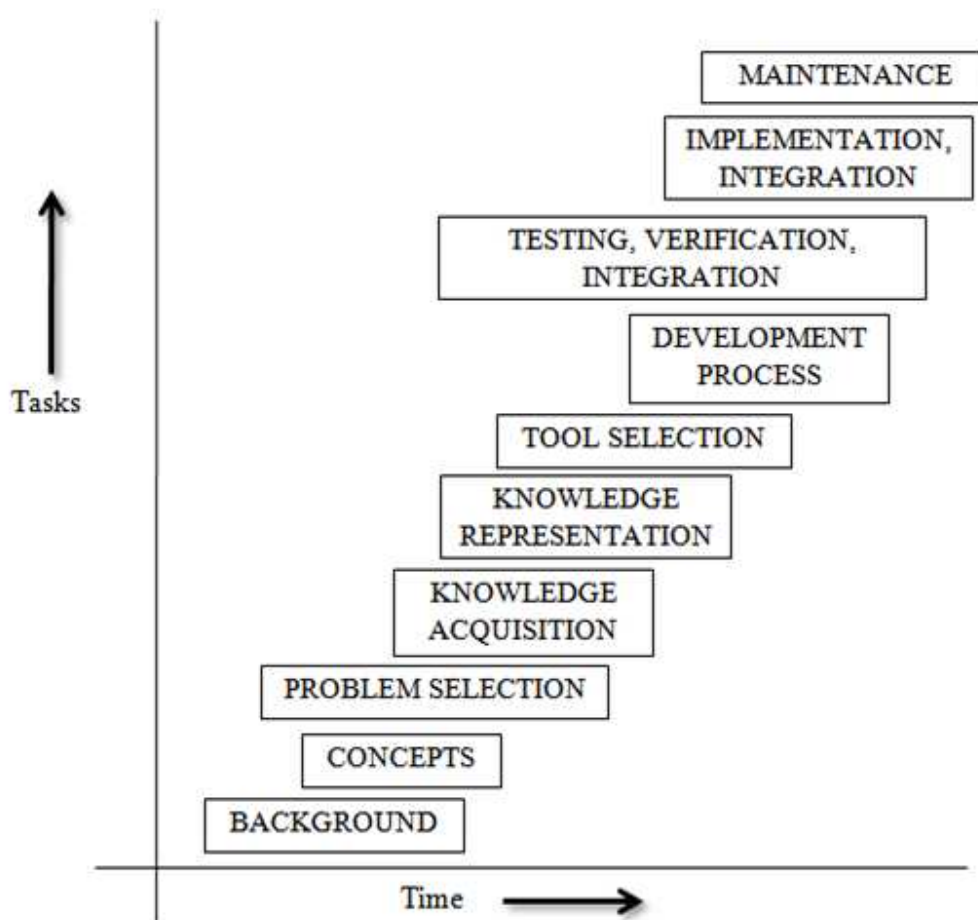


Figure Hierarchy of expert system development process

Some latest developments in the expert system area are as follows:

1. Availability of many tools that are designed to expedite the construction of expert system at a reduced cost.

2. Increased use of expert systems in many tasks ranging from help desks to complex military and space shuttle applications.
3. Use of multiple knowledge bases.
4. Improvements in knowledge acquisition.
5. Use of the internet to disseminate software and expertise.
6. Increased use of object oriented programming approach in knowledge representation.
7. The multiple use of heuristic knowledge in several applications.
8. Enables the user to think about hypothetical reasoning.

CHARACTERISTICS OF AN EXPERT SYSTEM

The growth of expert system is expected to continue for several years. With the continuing growth, many new and exciting applications will emerge. An expert system operates as an interactive system that responds to questions, asks for clarification, makes recommendations and generally aids the decision making process. Expert system provides expert advice and guidance in a wide variety of activities from computer diagnosis to delicate medical surgery.

An expert system is usually designed to have the following general characteristics.

1. **High level Performance:** The system must be capable of responding at a level of competency equal to or better than an expert system in the field. The quality of the advice given by the system should be in a high level integrity and for which the performance ratio should be also very high.
2. **Domain Specificity:** Expert systems are typically very domain specific. For ex., a diagnostic expert system for troubleshooting computers must actually perform all the necessary data manipulation as a human expert would. The developer of such a system must limit his or her scope of the system to just what is needed to solve the target problem. Special tools or programming languages are often needed to accomplish the specific objectives of the system.
3. **Good Reliability:** The expert system must be as reliable as a human expert.
4. **Understandable:** The system should be understandable i.e. be able to explain the steps of reasoning while executing. The expert system should have an explanation capability similar to the reasoning ability of human experts.
5. **Adequate Response time:** The system should be designed in such a way that it is able to perform within a small amount of time, comparable to or better than the time taken by a human expert to reach at a decision point. An expert system that takes a year to reach a decision compared to a human expert's time of one hour would not be useful.
6. **Use symbolic representations:** Expert system use symbolic representations for knowledge (rules, networks or frames) and perform their inference through symbolic computations that closely resemble manipulations of natural language.

7. **Linked with Metaknowledge:** Expert systems often reason with metaknowledge i.e. they reason with knowledge about themselves and their own knowledge limits and capabilities. The use of metaknowledge is quite interactive and simple for various data representations.
8. **Expertise knowledge:** Real experts not only produce good solutions but also find them quickly. So, an expert system must be skillful in applying its knowledge to produce solutions both efficiently and effectively by using the intelligence human experts.
9. **Justified Reasoning:** This allows the users to ask the expert system to justify the solution or advice provided by it. Normally, expert systems justify their answers or advice by explaining their reasoning. If a system is a rule based system, it provides to the user all the rules and facts it has used to achieve its answer.
10. **Explaining capability:** Expert systems are capable of explaining how a particular conclusion was reached and why requested information is needed during a consultation. This is very important as it gives the user a chance to access and understand the system's reasoning ability, thereby improving the user's confidence in the system.
11. **Special Programming Languages:** Expert systems are typically written in special programming languages. The use of languages like LISP and PROLOG in the development of an expert system simplifies the coding process. The major advantage of these languages, as compared to conventional programming languages is the simplicity of the addition, elimination or substitution of new rules and memory management capabilities. Some of the distinguishing characteristics of programming languages needed for expert system work are as follows:
 - a) Efficient mix of integer and real variables.
 - b) Good memory management procedures.
 - c) Extensive data manipulation routines.
 - d) Incremental compilation.
 - e) Tagged memory architecture.
 - f) Efficient search procedures.
 - g) Optimization of the systems environment.

STRUCTURE OF AN EXPERT SYSTEM

The structure of expert systems reflect the knowledge engineers understanding of the methods of representing knowledge and of how to perform intelligent decision making tasks with the support of a computer based system. Complex decisions involve intricate combination of factual and heuristic knowledge. In order for the computer to be able to retrieve and effectively use heuristic knowledge, the knowledge must be organized in an easily accessible format that distinguishes among data, knowledge and control structures. For this reason expert systems are organized in three distinct levels like:

- a) **Knowledge Base:** It consists of problem solving rules, procedures and intrinsic data relevant to the problem domain. The knowledge base constitutes the problem solving rules, facts or intuition that a

human expert might use in solving problems in a given problem domain. The knowledge base is usually stored in terms of if-then rules. The working memory represents relevant data for the current problem being solved.

- b) **Working Memory:** It refers to task specific data for the problem under consideration. This is the dynamic module of the system. It consists of an essential component called database. In general, the workspace contains a set called rule base, i.e. it contains a set of rules that to be used by a system at a given moment.
- c) **Inference Engine:** This is a generic control mechanism that applies the axiomatic knowledge in the knowledge base to the task specific data to arrive at some solution or conclusion. Inference in production systems is accomplished by a process of chaining through the rules recursively, either in a forward or in a backward direction until a conclusion is reached.

These three pieces may very well come from different sources. The inference engine, such as VP-Expert, may come from a commercial vendor. The knowledge base may be a specific diagnostic knowledge base compiled by a consulting firm, and the problem data may be supplied by the end user. A knowledge base is the nucleus of the expert system structure. A knowledge base is created by knowledge engineers, who translate the knowledge of real human experts into rules and strategies. These rules and strategies can change depending on the prevailing problem scenario. The knowledge base provides the expert system with the capability to recommend directions for user inquiry. The system also instigates further investigation into areas that may be important to a certain line of reasoning but not apparent to the user. The general structure of an expert system is given in figure .

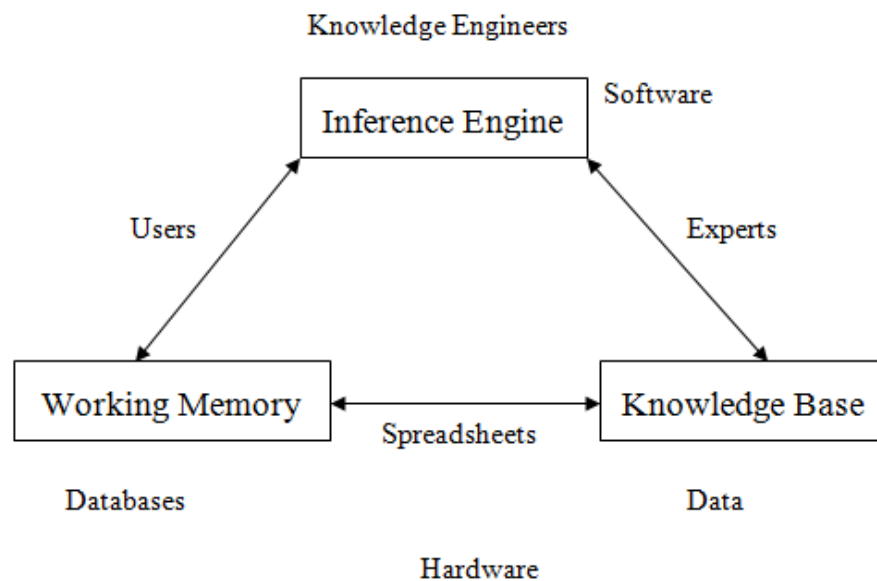


Figure Expert Systems Organization and Operating Environment The

modularity of an expert system is an important distinguishing characteristics compared to a conventional computer program. Modularity is affected in an expert system by the use of three distinct components as shown in fig 6.2. A good expert system is expected to grow as it learns from user feedback. Feedback is incorporated into the knowledge base as appropriate to make the expert system smarter. The dynamism of the application environment for expert systems is based on the individual dynamism of the components. This can be classified into three categories as follows.

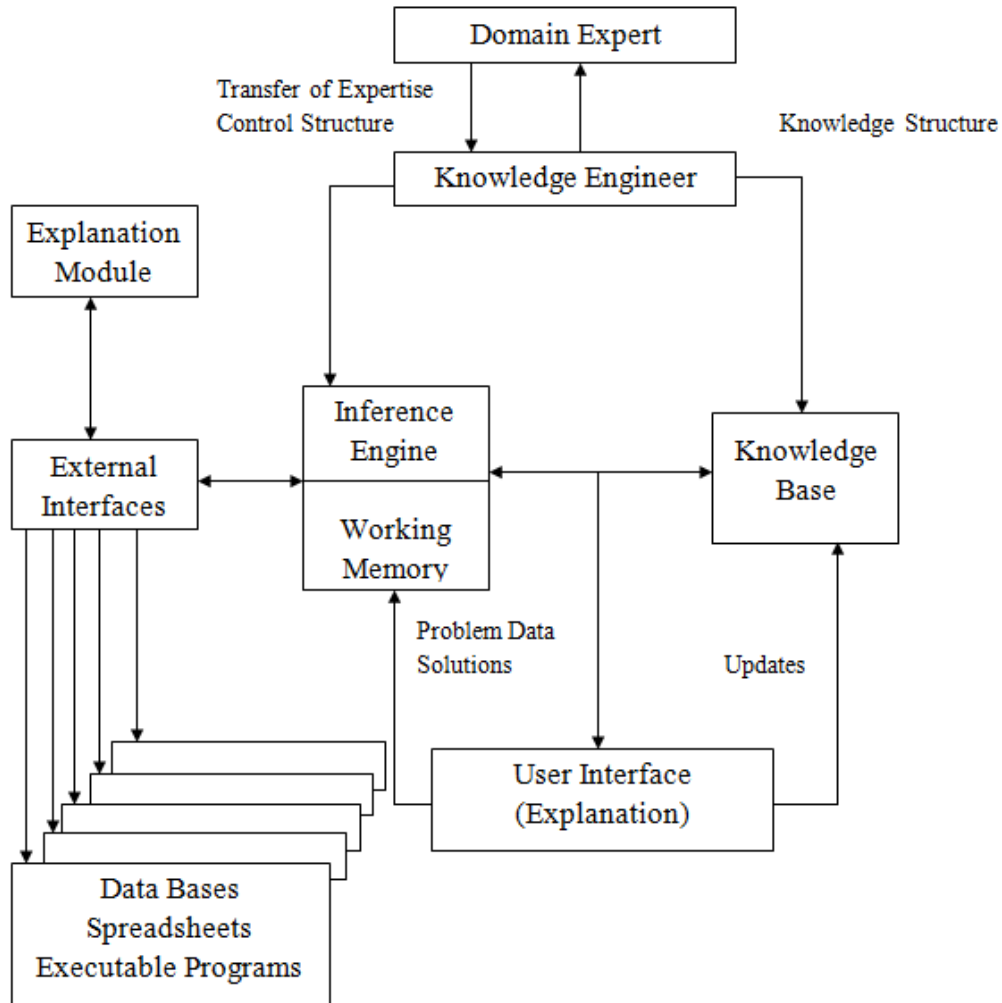
- a) **Most dynamic:** The most dynamic part of an expert system is always the working memory. The content of the working memory, sometimes called the data structure, changes with each problem situation. Consequently, it is the most dynamic component of an expert system assuming, of course that it is kept current.
- b) **Moderately dynamic:** This part in the expert system is the knowledge base. The knowledge base need not change unless a new piece of information arises that indicates a change in the problem solution procedure. Changes in the knowledge base should be carefully evaluated before being implemented. In effect, changes should not be based on just one consultation experience.
- c) **Least dynamic:** The least dynamic part is the inference engine. As the control and coding structure of an inference engine is very strict, so changes are made only if absolutely necessary to correct a bug or enhance the inferential process. Commercial inference engines, in particular, change only at the discretion of the developer. Since frequent updates can be disruptive and costly to clients, most commercial software developers try to minimize the frequency of updates.

RULE BASED ARCHITECTURE OF AN EXPERT SYSTEM

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

Then: Take action4

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



Integration of Expert systems Components

The components of the rule based architecture are as follows.

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

- a) To get answer of his/her queries.
- b) To get explanation about the solution for psychological satisfaction.

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

2. **Explanation Module:** The explanation module explains the reasoning of the system to a user. It provides the user with an explanation of the reasoning process when requested. The credibility of

expert system will be established only when it is able to explain “how and why” a particular conclusion is drawn. This explanation increases the belief of user in the expert system.

- a) **Explanation(How):** To respond to a how query, the explanation module traces the chain of rules fired during a consultation with the user. This explanation mode can be activated once the process is over. It explains how a fact was deduced by the system and similarly how a rule was/wasn't used. The simplest way to specify this is to explain the rule which allows the deduction. For e.g. If the system (S) will give information about the parent-child relationship to the user (U) then the followings can be possible.

S: My diagnosis is “A is the father of B”

U: How?

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

Fact no 11: A is the parent of Hari.

Fact no 15: A is a male.

Fact no 110: X is father of Y:

X is parent of Y, X is male.

So A is the father of B.

- b) **Explanation (Why)?** To respond to a why query, the explanation module must be able to explain why certain information is needed by the inference engine to complete a step in the reasoning process. This mode of explanation is dynamic and is used when the system functions in backward chaining. If the user doesn't understand the reason behind the question. The system is capable of explaining why the question was asked.

For example S: Is the following true?

A is the father of B.

U: Why?

S: I need the fact:

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

By using the rule no. 4:

A is the father of B:

B is the son of A.

3. **Working Memory:** It is a global database of facts used by the rules.
4. **Knowledge Engineering:** The primary people involved in building an expert system are the knowledge engineer, the domain expert and the end user. Once the knowledge engineer has obtained a general overview of the problem domain and gone through several problem solving sessions with

the domain expert, he/she is ready to begin actually designing the system, selecting a way to represent the knowledge, determining the search strategy (backward or forward) and designing the user interface. After making complete designs, the knowledge engineer builds a prototype. The prototype should be able to solve problems in a small area of the domain. Once the prototype has been implemented, the knowledge engineer and domain expert test and refine its knowledge by giving it problems to solve and correcting its disadvantages.

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

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

APPLICATIONS OF EXPERT SYTEM

There are several major application areas of expert system such as agriculture, education, environment, law manufacturing, medicine power system etc. Expert system is used to develop a large number of new products as well as new configurations of established products. When established products are modified to include an expert system as a component or when an established product item is replaced with an expert system, the expert system supported entity is called intelligent. Expert systems are designed and created to facilitate tasks in the fields of accounting, medicine, process control, financial service, production, education etc. The foundation of a successful expert system depends on a series of technical procedures and development that may be designed by certain related experts.

Expert Systems are for everyone

Everyone can find an application potential in the field of expert systems. Contrary to the belief that expert systems may pose a threat to job security, expert systems can actually help to create opportunities for new job areas. No matter which is of business one is engages in, expert systems can fulfill the need for higher productivity and reliability of decisions. Some job opportunities offered by the expert system are listed below:

- Basic Research
- Applied Research
- Knowledge Engineering
- The development of Inference engine
- Training
- Sales and marketing

Expert System in Education

In the field of education, many of the expert system's application are embedded inside the Intelligent Tutoring System (ITS) by using techniques from adaptive hypertext and hypermedia. Most of the system usually will assist student in their learning by using adaptation techniques to personalize with the environment prior knowledge of student and student's ability to learn. Expert system in education has expanded very consistently from micro computer to web based and agent based technology. Web based expert system can provide an excellent alternative to private tutoring at any time from any place where internet is provided. Agent based expert system will help users by finding materials from the web based on the user's profile. Expert system also had tremendous changes in the applying of methods and techniques. Expert system are beneficial as a teaching tools because it has equipped with the unique features which allow users to ask question on how, why and what format. When it is used in the class environment, surely it will give many benefit to student as it prepare the answer without referring to the teacher. Beside that, expert system is able to give reasons towards the given answer. Expert system had been used in several fields of study including computer animation, computer science and engineering, language teaching business study etc.

Expert system in Agriculture

The expert system for agriculture is same as like other fields. Here also the expert system uses the rule based structure and the knowledge of a human expert is captured in the form of IF-THEN rules and facts which are used to solve problems by answering questions typed at a keyboard attached to a computer. For example, in pest control, the need to spray, selection of a chemical to spray, mixing and application etc. The early, state of developing the expert systems are in the 1960's and 1970's were typically written on a mainframe computer in the programming language based on LISP. Some examples of these expert

systems are MACSYMA developed at the Massachusetts Institute of Technology (MIT) for assisting individuals in solving complex mathematical problems. Other examples may be MYCIN, DENDRAL, and CALEX etc. The rises of the agricultural expert system are to help the farmers to do single point decisions, which to have a well planning for before start to do anything on their land. It is used to design an irrigation system for their plantation use. Also some of the other functions of agricultural expert system are:

- To predict the extreme events such as thunderstorms and frost.
- To select the most suitable crop variety.
- Diagnosis of liver stock disorder and many more.

Expert System for a particular decision problem

The expert system can be used as a stand alone advisory system for the specific knowledge domain. It also can provide decision support for a high level human expert. The main purposes, the rises of the expert system are as a delivery system for extension information, to provide management education for decision makers and for dissemination of up-to-date scientific information in a readily accessible and easily understood form, to agricultural researchers, advisers and farmers. By the help of an expert system, the farmers can produce a more high quality product to the citizen.

Expert System for Text Animation (ESTA)

The idea behind creating an expert system is that it can enable many people to benefit from the knowledge of one person – the expert. By providing it with a knowledge base for a certain subject area, ESTA can be used to create an expert system for the subject:

$$\text{ESTA} + \text{Knowledge base} = \text{Expert System}$$

Each knowledge base contains rules for a specific domain. A knowledge base for an expert system to give tax advice might contain rules relating marital status, mortgage commitments and age to the advisability of taking out a new life insurance policy. ESTA has all facilities to write the rules that will make up a knowledge base. ESTA has an inference engine which can use the rules in the knowledge base to determine which advice is to be given to the expert system user. ESTA also features the ability for the expert system user to obtain answers to questions such as “how” and “why”. ESTA is used by a knowledge engineer to create a knowledge base and by the expert system user to consult a knowledge base. Knowledge representation in ESTA is based on the items like sections, parameters, title.