
Prof.Ajay Pashankar
Department Of Computer Science & Information Technology
ajaypashankar7
Prof.Ajay Pashankar
www.profajaypashankar.com

MSc IT PART II
SEM III(AI TRACK)

APPLIED ARTIFICIAL INTELLIGENCE NOTES

BY: PROF.AJAY PASHANKAR

Unit	*Details*	*Lectures*	*Outcome*
I	**Review of AI:** History, foundation and Applications **Expert System and Applications:** Phases in Building Expert System, Expert System Architecture, Expert System versus Traditional Systems, Rule based Expert Systems, Blackboard Systems, Truth Maintenance System, Application of Expert Systems, Shells and Tools	**12**	**CO1**
II	**Probability Theory:** joint probability, conditional probability, Bayes's theorem, probabilities in rules and facts of rule based system, cumulative probabilities, rule based system and Bayesian method **Fuzzy Sets and Fuzzy Logic:** Fuzzy Sets, Fuzzy set operations, Types of Member ship Functions, Multivalued Logic, Fuzzy Logic, Linguistic variables and Hedges, Fuzzy propositions, inference rules for fuzzy propositions, fuzzy systems, possibility theory and other enhancement to Logic	**12**	**CO2**
III	**Machine Learning Paradigms:** Machine Learning systems, supervised and un-supervised learning, inductive learning, deductive learning, clustering, support vector machines, cased based reasoning and learning. **Artificial Neural Networks:** Artificial Neural Networks, Single-Layer feedforward networks, multi-layer feed-forward networks, radial basis function networks, design issues of artificial neural networks and recurrent networks	**12**	**CO3**
IV	**Evolutionary Computation:** Soft computing, genetic algorithms, genetic programming concepts, evolutionary programming, swarm intelligence, ant colony paradigm, particle swarm optimization and applications of evolutionary algorithms. **Intelligent Agents:** Agents vs software programs, classification of agents, working of an agent, single agent and multiagent systems, performance evaluation, architecture, agent communication language, applications	**12**	**CO4**
V	Advanced Knowledge Representation Techniques: Conceptual dependency theory, script structures, CYC theory, script structure, CYC theory, case grammars, semantic web. Natural Language Processing: Sentence Analysis phases, grammars and parsers, types of parsers, semantic analysis, universal networking language, dictionary	**12**	**CO5**

CHAPTER I : REVIEW OF AI

Topics Covered: History, foundation and Applications

What is Artificial Intelligence?

It is a branch of Computer Science that pursues creating the computers or machines as intelligentas human beings.

It is the science and engineering of making intelligent machines, especially intelligent computer programs.

It is related to the similar task of using computers to understand human intelligence, but **AI** does not have to confine itself to methods that are biologically observable

Definition: Artificial Intelligence is the study of how to make computers do things, which, at themoment, people do better.

According to the father of Artificial Intelligence, John McCarthy, it is *"The science and engineering of making intelligent machines, especially intelligent computer programs".*

Artificial Intelligence is a way of **making a computer, a computer-controlled robot, or a software think intelligently**, in the similar manner the intelligent humans think.

AI is accomplished by studying how human brain thinks and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

It has gained prominence recently due, in part, to big data, or the increase in speed, size and variety of data businesses are now collecting. AI can perform tasks such as identifying patterns in the data more efficiently than humans, enabling businesses to gain more insight out of their data.

From a **business** perspective AI is a set of very powerful tools, and methodologies for using those tools to solve business problems.

From a **programming** perspective, AI includes the study of symbolic programming, problem solving, and search.

AI Vocabulary

Intelligence relates to tasks involving higher mental processes, e.g. creativity, solving problems, pattern recognition, classification, learning, induction, deduction, building analogies, optimization, language processing, knowledge and many more. Intelligence is the computational part of the ability to achieve goals.

Intelligent behaviour is depicted by perceiving one's environment, acting in complex environments, learning and understanding from experience, reasoning to solve problems and discover hidden knowledge, applying knowledge successfully in new situations, thinking abstractly, using analogies, communicating with others and more.

Science based goals of AI pertain to developing concepts, mechanisms and understanding biological intelligent behaviour. The emphasis is on understanding intelligent behaviour.

Engineering based goals of AI relate to developing concepts, theory and practice of building intelligent machines. The emphasis is on system building.

AI Techniques depict how we represent, manipulate and reason with knowledge in order to solve problems. Knowledge is a collection of 'facts'. To manipulate these facts by a program, a suitable representation is required. A good representation facilitates problem solving.

Learning means that programs learn from what facts or behaviour can represent. Learning denotes changes in the systems that are adaptive in other words, it enables the system to do the same task(s) more efficiently next time.

Applications of AI refers to problem solving, search and control strategies, speech recognition, natural language understanding, computer vision, expert systems, etc.

Problems of AI:

Intelligence does not imply perfect understanding; every intelligent being has limited perception, memory and computation. Many points on the spectrum of intelligence versus cost are viable, from insects to humans. AI seeks to understand the computations required from intelligent behaviour and to produce computer systems that exhibit intelligence. Aspects of intelligence studied by AI include perception, communicational using human languages, reasoning, planning, learning and memory.

The following questions are to be considered before we can step forward:

1. What are the underlying assumptions about intelligence?
2. What kinds of techniques will be useful for solving AI problems?
3. At what level human intelligence can be modelled?
4. When will it be realized when an intelligent program has been built?

Branches of AI:

A list of branches of AI is given below. However some branches are surely missing, because no one has identified them yet. Some of these may be regarded as concepts or topics rather than full branches.

Logical AI — In general the facts of the specific situation in which it must act, and its goals are all represented by sentences of some mathematical logical language. The program decides what to do by inferring that certain actions are appropriate for achieving its goals.

Search — Artificial Intelligence programs often examine large numbers of possibilities – for example, moves in a chess game and inferences by a theorem proving program. Discoveries are frequently made about how to do this more efficiently in various domains.

Pattern Recognition — When a program makes observations of some kind, it is often plannedto compare what it sees with a pattern. For example, a vision program may try to match a pattern of eyes and a nose in a scene in order to find a face. More complex patterns are like a natural language text, a chess position or in the history of some event. These more complex patterns require quite different methods than do the simple patterns that have been studied the most.

Representation — Usually languages of mathematical logic are used to represent the facts aboutthe world.

Inference — Others can be inferred from some facts. Mathematical logical deduction is sufficient for some purposes, but new methods of *non-monotonic* inference have been added to the logic since the 1970s. The simplest kind of non-monotonic reasoning is default reasoning in which a conclusion is to be inferred by default. But the conclusion can be withdrawn if there is evidence to the divergent. For example, when we hear of a bird, we infer that it can fly, but this conclusion can be reversed when we hear that it is a penguin. It is the possibility that a conclusion may have to be withdrawn that constitutes the non-monotonic character of the reasoning. Normal logical reasoning is monotonic, in that the set of conclusions can be drawn from a set of premises, i.e. monotonic increasing function of the premises. Circumscription is another form of non-monotonic reasoning.

Common sense knowledge and Reasoning — This is the area in which AI is farthest from the human level, in spite of the fact that it has been an active research area since the 1950s. While there has been considerable progress in developing systems of *non-monotonic reasoning* and theories of action, yet more new ideas are needed.

Learning from experience — There are some rules expressed in logic for learning. Programs can only learn what facts or behaviour their formalisms can represent, and unfortunately learning systems are almost all based on very limited abilities to represent information.

Planning — Planning starts with general facts about the world (especially facts about the effects of actions), facts about the particular situation and a statement of a goal. From these, planning programs generate a strategy for achieving the goal. In the most common cases, the strategy is just a sequence of actions.

Epistemology — This is a study of the kinds of knowledge that are required for solving problems in the world.

Ontology — Ontology is the study of the kinds of things that exist. In AI the programs and sentences deal with various kinds of objects and we study what these kinds are and what their basic properties are. Ontology assumed importance from the 1990s.

Heuristics — A heuristic is a way of trying to discover something or an idea embedded in a program. The term is used variously in AI. *Heuristic functions* are used in some approaches to search or to measure how far a node in a search tree seems to be from a goal. *Heuristic predicates* that compare two nodes in a search tree to see if one is better than the other, i.e. constitutes an advance toward the goal, and may be more useful.

Genetic programming — Genetic programming is an automated method for creating a working computer program from a high-level problem statement of a problem. Genetic programming starts from a high-level statement of 'what needs to be done' and automatically creates acomputer program to solve the problem.

Applications of AI

AI has applications in all fields of human study, such as finance and economics, environmentalengineering, chemistry, computer science, and so on. Some of the applications of AI are listed below:

- ☐ Perception
  - ■ Machine vision
  - ■ Speech understanding
  - ■ Touch (*tactile* or *haptic*) sensation
- ☐ Robotics
- ☐ Natural Language Processing
  - ■ Natural Language Understanding
  - ■ Speech Understanding
  - ■ Language Generation
  - ■ Machine Translation
- ☐ Planning
- ☐ Expert Systems
- ☐ Machine Learning
- ☐ Theorem Proving
- ☐ Symbolic Mathematics
- ☐ Game Playing

AI Technique:

Artificial Intelligence research during the last three decades has concluded that *Intelligence requires knowledge*. To compensate overwhelming quality, knowledge possesses less desirableproperties.

A. It is huge.
B. It is difficult to characterize correctly.
C. It is constantly varying.
D. It differs from data by being organized in a way that corresponds to its application.
E. It is complicated.

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

- The knowledge captures generalizations that share properties, are groupedtogether, rather than being allowed separate representation.
- It can be understood by people who must provide it—even though for manyprograms bulk of the data comes automatically from readings.
- In many AI domains, how the people understand the same people must supply theknowledge to a 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 rangeof possibilities that must be usually considered.

In order to characterize an AI technique let us consider initially OXO or tic-tac-toe and use a series of different approaches to play the game.

The programs increase in complexity, their use of generalizations, the clarity of their knowledge and the extensibility of their approach. In this way they move towards being representations of AI techniques.

Example-1: Tic-Tac-Toe

1.1 The first approach (simple)

The Tic-Tac-Toe game consists of a nine element vector called BOARD; it represents the numbers 1 to 9 in three rows.

1	2	3
4	5	6
7	8	9

An element contains the value 0 for blank, 1 for X and 2 for O. A MOVETABLE vector consists of 19,683 elements (3^9) and is needed where each element is a nine element vector. The contents of the vector are especially chosen to help the algorithm.

The algorithm makes moves by pursuing the following:

1. View the vector as a ternary number. Convert it to a decimal number.
2. Use the decimal number as an index in MOVETABLE and access the vector.
3. Set BOARD to this vector indicating how the board looks after the move. This approach is capable in time but it has several disadvantages. It takes more space and requires stunning

effort to calculate the decimal numbers. This method is specific to this game and cannot becompleted.

1.2 The second approach

The structure of the data is as before but we use 2 for a blank, 3 for an X and 5 for an O. A variable called TURN indicates 1 for the first move and 9 for the last. The algorithm consistsof three actions:

MAKE2 which returns 5 if the centre square is blank; otherwise it returns any blank non- corner square, i.e. 2, 4, 6 or 8. POSSWIN (p) returns 0 if player p cannot win on the next move and otherwise returns the number of the square that gives a winning move.

It checks each line using products 3*3*2 = 18 gives a win for X, 5*5*2=50 gives a win for O, and the winning move is the holder of the blank. GO (n) makes a move to square n setting BOARD[n] to 3 or 5.

This algorithm is more involved and takes longer but it is more efficient in storage which compensates for its longer time. It depends on the programmer's skill.

1.3 The final approach

The structure of the data consists of BOARD which contains a nine element vector, a list of board positions that could result from the next move and a number representing an estimation of howthe board position leads to an ultimate win for the player to move.

This algorithm looks ahead to make a decision on the next move by deciding which the most promising move or the most suitable move at any stage would be and selects the same.

Consider all possible moves and replies that the program can make. Continue this process for as long as time permits until a winner emerges, and then choose the move that leads to the computer program winning, if possible in the shortest time.

Actually this is most difficult to program by a good limit but it is as far that the technique canbe extended to in any game. This method makes relatively fewer loads on the programmer in termsof the game technique but the overall game strategy must be known to the adviser.

CHAPTER II EXPERT SYSTEM AND APPLICATIONS:

Topics Covered:

Phases in Building Expert System, Expert System Architecture, Expert System versus Traditional Systems, Rule based Expert Systems, Blackboard Systems, Truth Maintenance System, Application of Expert Systems, Shells and Tools

---

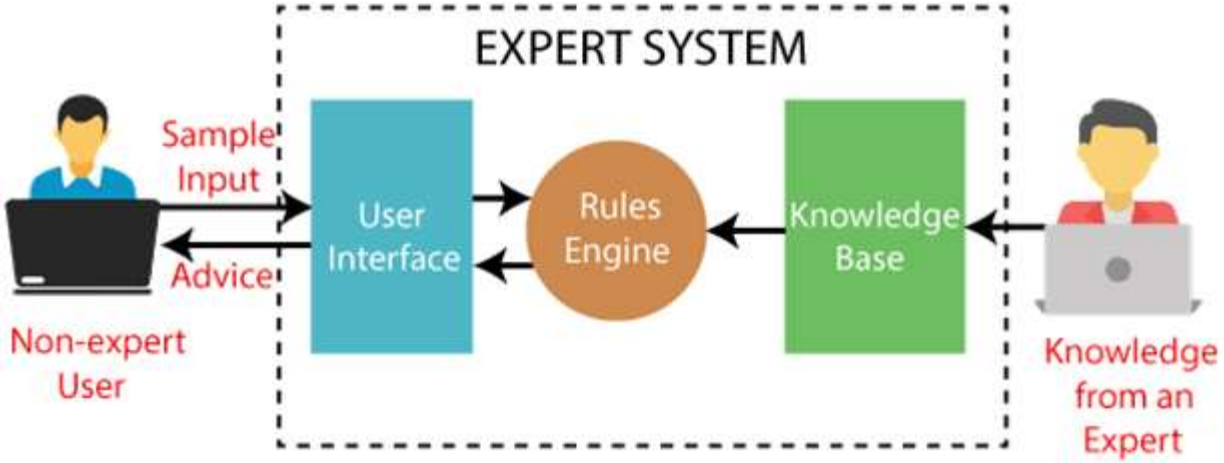
What is an Expert System?

An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

The expert system is a part of AI, and the first ES was developed in the year 1970, which was the first successful approach of artificial intelligence. It solves the most complex issue as an expert by extracting the knowledge stored in its knowledge base. The system helps in decision making for compsex problems using **both facts and heuristics like a human expert**. It is called so because it contains the expert knowledge of a specific domain and can solve any complex problem of that particular domain. These systems are designed for a specific domain, such as **medicine, science,** etc.

The performance of an expert system is based on the expert's knowledge stored in its knowledge base. The more knowledge stored in the KB, the more that system improves its performance. One of the common examples of an ES is a suggestion of spelling errors while typing in the Google search box.

Below is the block diagram that represents the working of an expert system:



Note: It is important to remember that an expert system is not used to replace the human experts; instead, it is used to assist the human in making a complex decision. These systems do not have human capabilities of thinking and work on the basis of the knowledge base of the particular domain.

Below are some popular examples of the Expert System:

- **DENDRAL:** It was an artificial intelligence project that was made as a chemical analysis expert system. It was used in organic chemistry to detect unknown organic molecules with the help of their mass spectra and knowledge base of chemistry.
- **MYCIN:** It was one of the earliest backward chaining expert systems that was designed to find the bacteria causing infections like bacteraemia and meningitis. It was also used for the recommendation of antibiotics and the diagnosis of blood clotting diseases.
- **PXDES:** It is an expert system that is used to determine the type and level of lung cancer. To determine the disease, it takes a picture from the upper body, which looks like the shadow. This shadow identifies the type and degree of harm.
- **CaDeT:** The CaDet expert system is a diagnostic support system that can detect cancer at early stages.

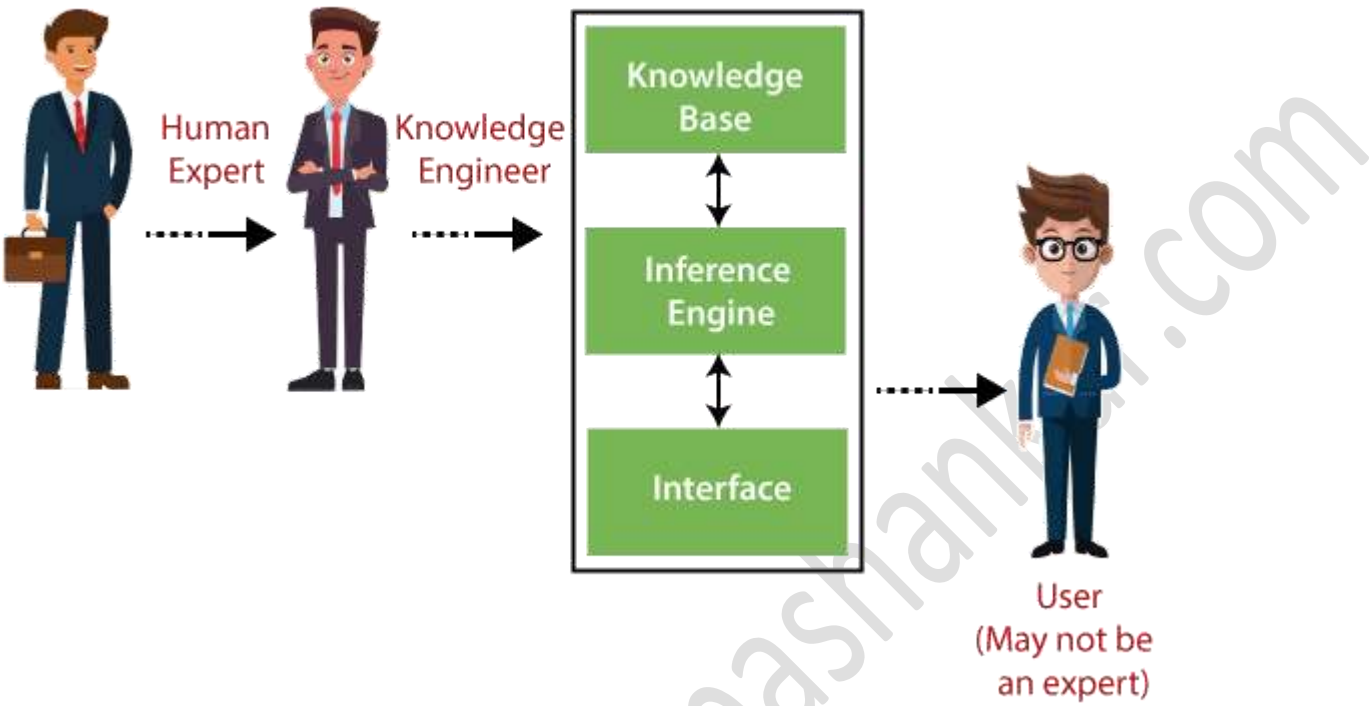
Characteristics of Expert System

- **High Performance:** The expert system provides high performance for solving any type of complex problem of a specific domain with high efficiency and accuracy.
- **Understandable:** It responds in a way that can be easily understandable by the user. It can take input in human language and provides the output in the same way.
- **Reliable:** It is much reliable for generating an efficient and accurate output.
- **Highly responsive:** ES provides the result for any complex query within a very short period of time.
-

Components of Expert System

An expert system mainly consists of three components:

- **User Interface**
- **Inference Engine**
- **Knowledge Base**



1. **User Interface**

With the help of a user interface, the expert system interacts with the user, takes queries as an input in a readable format, and passes it to the inference engine. After getting the response from the inference engine, it displays the output to the user. In other words, **it is an interface that helps a non-expert user to communicate with the expert system to find a solution**.

2. **Inference Engine(Rules of Engine)**

- The inference engine is known as the brain of the expert system as it is the main processing unit of the system. It applies inference rules to the knowledge base to derive a conclusion or deduce new information. It helps in deriving an error-free solution of queries asked by the user.
- With the help of an inference engine, the system extracts the knowledge from the knowledge base.
- There are two types of inference engine:
- **Deterministic Inference engine:** The conclusions drawn from this type of inference engine are assumed to be true. It is based on **facts** and **rules**.
- **Probabilistic Inference engine:** This type of inference engine contains uncertainty in conclusions, and based on the probability.

Inference engine uses the below modes to derive the solutions:

- **Forward Chaining:** It starts from the known facts and rules, and applies the inference rules to add their conclusion to the known facts.
- **Backward Chaining:** It is a backward reasoning method that starts from the goal and works backward to prove the known facts.

3. **Knowledge Base**

- The knowledgebase is a type of storage that stores knowledge acquired from the different experts of the particular domain. It is considered as big storage of knowledge. The more the knowledge base, the more precise will be the Expert System.
- It is similar to a database that contains information and rules of a particular domain or subject.
- One can also view the knowledge base as collections of objects and their attributes. Such as a Lion is an object and its attributes are it is a mammal, it is not a domestic animal, etc.

Components of Knowledge Base

- **Factual Knowledge:** The knowledge which is based on facts and accepted by knowledge engineers comes under factual knowledge.

- **Heuristic Knowledge:** This knowledge is based on practice, the ability to guess, evaluation, and experiences.

Knowledge Representation: It is used to formalize the knowledge stored in the knowledge base using the If-else rules.

Knowledge Acquisitions: It is the process of extracting, organizing, and structuring the domain knowledge, specifying the rules to acquire the knowledge from various experts, and store that knowledge into the knowledge base.

Development of Expert System

Here, we will explain the working of an expert system by taking an example of MYCIN ES. Below are some steps to build an MYCIN:

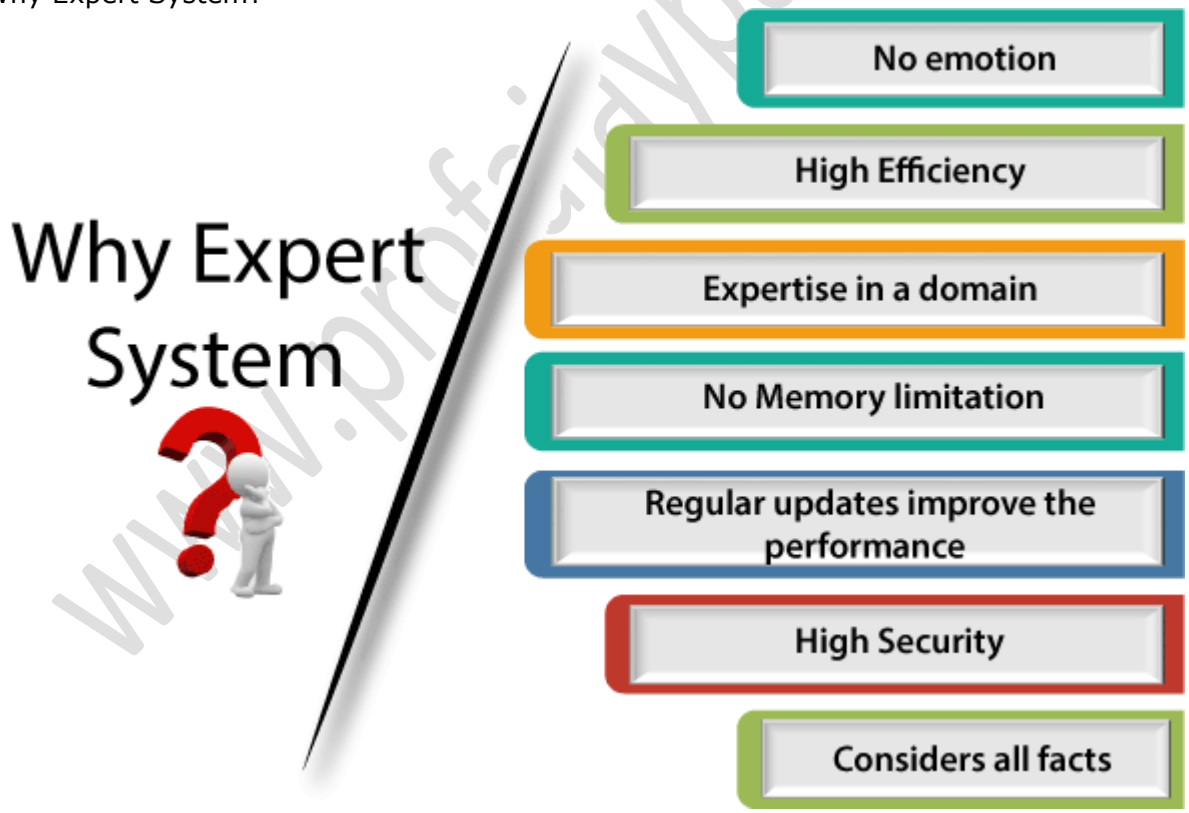
- Firstly, ES should be fed with expert knowledge. In the case of MYCIN, human experts specialized in the medical field of bacterial infection, provide information about the causes, symptoms, and other knowledge in that domain.
- The KB of the MYCIN is updated successfully. In order to test it, the doctor provides a new problem to it. The problem is to identify the presence of the bacteria by inputting the details of a patient, including the symptoms, current condition, and medical history.
- The ES will need a questionnaire to be filled by the patient to know the general information about the patient, such as gender, age, etc.
- Now the system has collected all the information, so it will find the solution for the problem by applying if-then rules using the inference engine and using the facts stored within the KB.
- In the end, it will provide a response to the patient by using the user interface.

Participants in the development of Expert System

There are three primary participants in the building of Expert System:

1. **Expert:** The success of an ES much depends on the knowledge provided by human experts. These experts are those persons who are specialized in that specific domain.
2. **Knowledge Engineer:** Knowledge engineer is the person who gathers the knowledge from the domain experts and then codifies that knowledge to the system according to the formalism.
3. **End-User:** This is a particular person or a group of people who may not be experts, and working on the expert system needs the solution or advice for his queries, which are complex.

Why Expert System?



Before using any technology, we must have an idea about why to use that technology and hence the same for the ES. Although we have human experts in every field, then what is the need to develop a computer-based system. So below are the points that are describing the need of the ES:

1. **No memory Limitations:** It can store as much data as required and can memorize it at the time of its application. But for human experts, there are some limitations to memorize all things at every time.
2. **High Efficiency:** If the knowledge base is updated with the correct knowledge, then it provides a highly efficient output, which may not be possible for a human.
3. **Expertise in a domain:** There are lots of human experts in each domain, and they all have different skills, different experiences, and different skills, so it is not easy to get a final output for the query. But if we put the knowledge gained from human experts into the expert system, then it provides an efficient output by mixing all the facts and knowledge
4. **Not affected by emotions:** These systems are not affected by human emotions such as fatigue, anger, depression, anxiety, etc.. Hence the performance remains constant.
5. **High security:** These systems provide high security to resolve any query.
6. **Considers all the facts:** To respond to any query, it checks and considers all the available facts and provides the result accordingly. But it is possible that a human expert may not consider some facts due to any reason.
7. **Regular updates improve the performance:** If there is an issue in the result provided by the expert systems, we can improve the performance of the system by updating the knowledge base.

Capabilities of the Expert System

Below are some capabilities of an Expert System:

- **Advising:** It is capable of advising the human being for the query of any domain from the particular ES.
- **Provide decision-making capabilities:** It provides the capability of decision making in any domain, such as for making any financial decision, decisions in medical science, etc.
- **Demonstrate a device:** It is capable of demonstrating any new products such as its features, specifications, how to use that product, etc.
- **Problem-solving:** It has problem-solving capabilities.
- **Explaining a problem:** It is also capable of providing a detailed description of an input problem.
- **Interpreting the input:** It is capable of interpreting the input given by the user.
- **Predicting results:** It can be used for the prediction of a result.
- **Diagnosis:** An ES designed for the medical field is capable of diagnosing a disease without using multiple components as it already contains various inbuilt medical tools.

Advantages of Expert System

- These systems are highly reproducible.
- They can be used for risky places where the human presence is not safe.
- Error possibilities are less if the KB contains correct knowledge.
- The performance of these systems remains steady as it is not affected by emotions, tension, or fatigue.
- They provide a very high speed to respond to a particular query.

Limitations of Expert System

- The response of the expert system may get wrong if the knowledge base contains the wrong information.
- Like a human being, it cannot produce a creative output for different scenarios.
- Its maintenance and development costs are very high.
- Knowledge acquisition for designing is much difficult.
- For each domain, we require a specific ES, which is one of the big limitations.
- It cannot learn from itself and hence requires manual updates.

Applications of Expert System

- **In designing and manufacturing domain**
  It can be broadly used for designing and manufacturing physical devices such as camera lenses and automobiles.
- **In the knowledge domain**
  These systems are primarily used for publishing the relevant knowledge to the users. The two popular ES used for this domain is an advisor and a tax advisor.
- **In the finance domain**
  In the finance industries, it is used to detect any type of possible fraud, suspicious activity, and advise bankers that if they should provide loans for business or not.
- **In the diagnosis and troubleshooting of devices**
  In medical diagnosis, the ES system is used, and it was the first area where these systems were used.

- **Planning and Scheduling**
  The expert systems can also be used for planning and scheduling some particular tasks for achieving the goal of that task.

Expert Systems (ES):

● Expert systems are knowledge based programs which provide expert quality solutions to the problems in specific domain of applications.
● The core components of expert system are – knowledge base and – navigational capability (inference engine)
● Generally its knowledge is extracted from human experts in the domain of application by knowledge Engineer. – Often based on useful thumb rules and experience rather than absolute certainties.
● A process of gathering knowledge from domain expert and codifying it according to the formalism is called knowledge engineering. 5.2 Phases in building Expert System
● There are different interdependent and overlapping phases in building an expert system as follows:
● Identification Phase: – Knowledge engineer finds out important features of the problem with the help of domain expert (human). – He tries to determine the type and scope of the problem, the kind of resources required, goal and objective of the ES.
● Conceptualization Phase: – In this phase, knowledge engineer and domain expert decide the concepts, relations and control mechanism needed to describe a problem solving.
Formalization Phase: – It involves expressing the key concepts and relations in some framework supported by ES building tools. – Formalized knowledge consists of data structures, inference rules, control strategies and languages for implementation.
● Implementation Phase: – During this phase, formalized knowledge is converted to working computer program initially called prototype of the whole system.
● Testing Phase: – It involves evaluating the performance and utility of prototype systems and revising it if need be. Domain expert evaluates the prototype system and his feedback help knowledge engineer to revise it.

5.3. Expert System Architecture

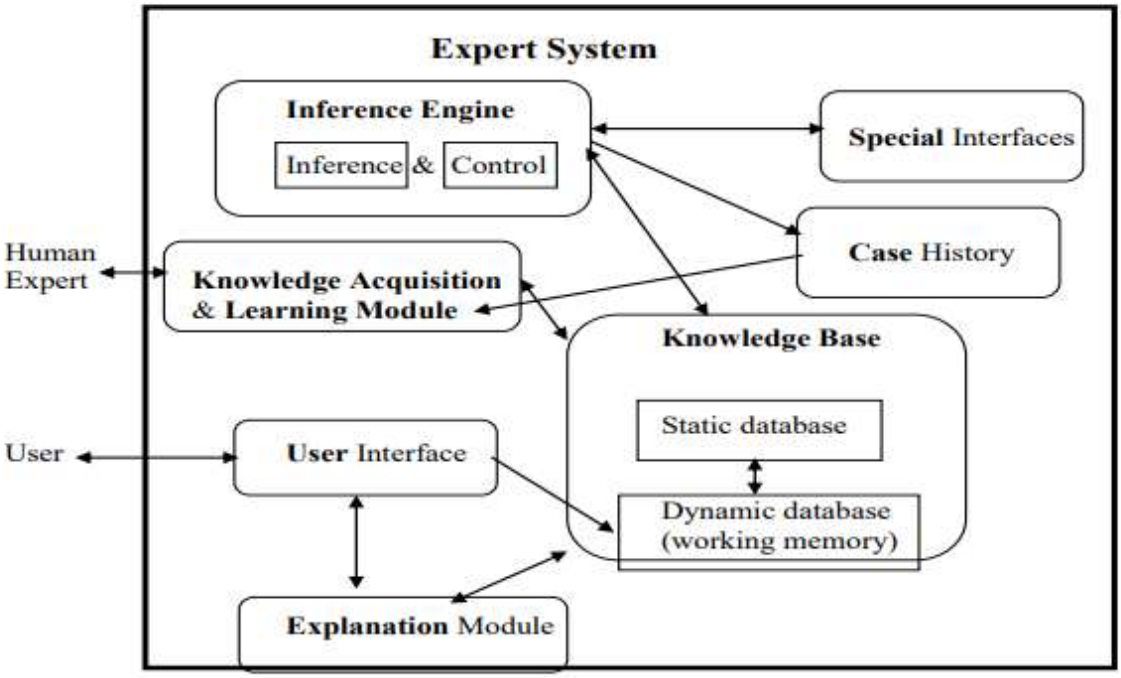


Fig: Architecture of Expert System

Knowledge Base (KB)

• KB consists of knowledge about problem domain in the form of static and dynamic databases.
• **Static knowledge consists** of – **rules and facts** which is complied as a part of the system and does not change during execution of the system.
• **Dynamic knowledge consists of facts related to a particular consultation of the system.** – At the beginning of the consultation, the dynamic knowledge base often called working memory is empty. –

As a consultation progresses, dynamic knowledge base grows and is used along with static knowledge in decision making. Working memory is deleted at the end of consultation of the system Inference Engine
• It consists of inference mechanism and control strategy.
• Inference means search through knowledge base and derive new knowledge.
• It involve formal reasoning involving matching and unification similar to the one performed by human expert to solve problems in a specific area of knowledge.
• Inference operates by using modus ponen rule.
• Control strategy determines the order in which rules are applied.
• There are mainly two types of control mechanism viz., forward chaining and backward chaining.

Knowledge Acquisition

• Knowledge acquisition module allows system to acquire knowledge about the problem domain.
• Sources of Knowledge for ES text books, reports, case studies, – empirical data and – domain expert experience.
• Updation of Knowledge can be done using knowledge acquisition module of the system. – insertion, – deletion and – updation of existing knowledge Case History
• Case History stores the file created by inference engine using the dynamic database created at the time of consultation.
• Useful for learning module to enrich its knowledge base.
• Different cases with solutions are stored in Case Base system.
• These cases are used for solving problem using Case Base Reasoning (CBR). Explanation module
• Most expert systems have explanation facilities that allow the user to ask the system why it asked some question, and how it reached to conclusion.
• It contains 'How' and 'Why' modules attached to it. – The sub-module 'How' tells the user about the process through w – hich system has reached to a particular solution – 'Why' sub-module tells that why is that particular solution offered.
• It explains user about the reasoning behind any particular problem solution.
• Questions are answered by referring to the system goals, the rules being used, and any existing problem data.

Explanation Modules Responses

User	Why module
System	Do you have running nose?
User	Yes.
System	I am trying to prove that you have cold. Do you have headache?
User	Yes
System	Do you have sore throat?
User	No
System	Then you do not have cold. Do you have fever?
User	Yes
System	Now I am looking at the possibility of measles. Do you have cough?
User	Yes
System	I can infer measles using rule "If symptoms are fever, cough, running_nose, then patient has measles" measles is concluded.
User	**How Module**
System	Since you have fever, running_nose and cough and there is a rule "If symptoms are fever, cough, running_nose, then patient has measles". So measles is concluded for you.

User Interfaces

- Allows user to communicate with system in interactive mode and helps system to create working knowledge for the problem to be solved.

Dialogue Module (User Interface)	
System	Do you have fever?
User	Yes
System	Do you have bad throat?
User	No
System	Do you have cough?
User	Yes
System	Are you suffering from running nose?
User	Yes
System	Are you suffering from headache?
User	No

Special interfaces

● It may be used for specialized activities such as handling uncertainty in knowledge.
● This is a major area of expert systems research that involves methods for reasoning with uncertain data and uncertain knowledge.
Knowledge is generally incomplete and uncertain.
● To deal with uncertain knowledge, a rule may have associated with it a confidence factor or a weight. ● The set of methods for using uncertain knowledge in combination with uncertain data in the reasoning process is called reasoning with uncertainty. 5.4 Rule Based Expert Systems
● A rule based expert system is one in which knowledge base is in the form of rules and facts. – Knowledge in the form of rules and facts is most popular way in designing expert systems.
● It is also called production system.
● Example: Suppose doctor gives a rule for measles as follows: "If symptoms are fever, cough, running_nose, rash and conjunctivitis then patient probably has measles".
● Prolog is most suitable for implementing such systems.

```
hypothesis(measles) :- symptom(fever), symptom(cough),
                       symptom(running_nose),symptom(conjunctivitis),
                       symptom(rash).
```

Simple Medical diagnostic system with dynamic databases:

● The system starts with consultation predicate, that initiates dialog with user to get information about various symptoms.
● Positive and negative symptoms are recorded in dynamic database and 'hypothesis(Disease)' is satisfied based on stored facts about symptoms.
● If the hypothesis goal is satisfied then the disease is displayed otherwise display 'sorry not able to diagnose'.
● Finally in both the situations, symptom database for a particular user is cleared. Query:?-consultation.

Medical Consultation System

```
consultation        :- writeln('Welcome to MC System'),
                       writeln('Input your name),
                       readln(Name),

                       hypothesis(Dis), !,

                       writeln(Name, 'probably has', Dis),
                       clear_consult_facts.

consultation        :- writeln('Sorry, not able to diagnose'),
                       clear_consult_facts

hypothesis(flu)     :- symptom(fever),
                       symptom(headache),

                       symptom(body_ache),
                       symptom(sore_throat),
                       symptom(cough),
                       symptom(chills),

                       symptom(running_nose),
                       symptom(conjunctivitis).

symptom(fever)      :- positive_ symp('Do you have
                       fever(y/n) ?', fever).
symptom(cough)      :- positive_ symp('Do you have
                       cough (y/n) ?', cough).

symptom(chills)     :- positive_ symp('Do you have
                       chills (y/n) ?', chills).

positive_ symp(_, X)  :- positive(X), !.

positive_ symp(Q, X)  :- not(negative(X)),
                         query(Q, X, R), R = 'y'.

query(Q, X, R)      :- writeln(Q), readln(R),
                       store(X, R).

store(X, 'y')       :- asserta(positive(X)).

store(X, 'n')       :- asserta(negative(X)).

clear_consult_facts :- retractall(positive(_)).

clear_consult_facts :- retractall(negative(_)).
```

Forward Chaining

● Prolog uses backward chaining as a control strategy, but forward chaining can be implemented in Prolog.

● In forward chaining, the facts from static and dynamic knowledge bases are taken and are used to test the rules through the process of unification.

● The rule is said to be fired and the conclusion (head of the rule) is added to the dynamic database when a rule succeeds.

● Prolog rules are coded as facts with two arguments, first argument be left side of rule and second is the list of sub goals in the right side of the rule.

● Represent prolog rule as a fact by rule_fact predicate and simple facts by fact predicate.

● Consider the following Prolog rules and facts with their corresponding new fact representations.

a:-b⇒rule_fact(a, [b]).

c:-b, e, f.⇒rule_fact(c, [b, e, f]).

b.⇒fact(b).

e.⇒fact(e).

f.⇒fact(f).

– Here a, b, c, e, f are atoms (predicates with arguments, if any). – Newly generated facts are stored in database file 'dfile' which is consulted in the prolog program. 5.6.Blackboard System – BS

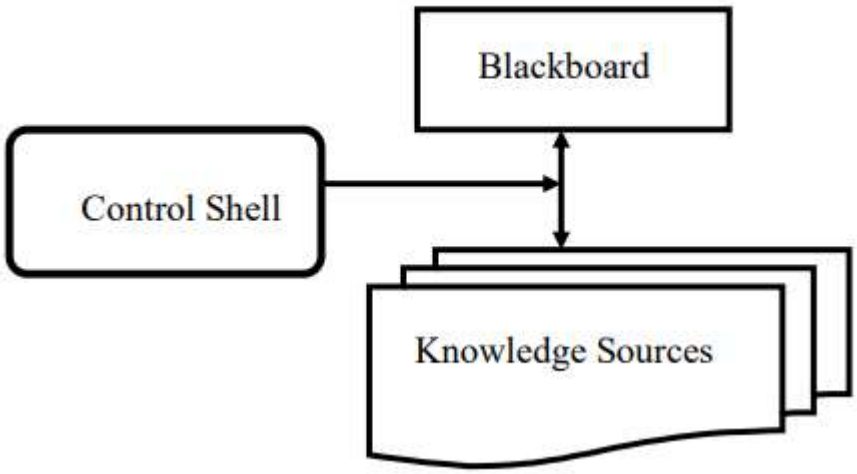
● Blackboard systems are used to capture dynamic environment with the help of domain experts.

BS uses a functional modularization of expertise knowledge in the form of Knowledge Sources (KS).

● KS are independent computational modules containing the expert knowledge needed to solve the problem. – Blackboard approach has an ability to integrate contributions dynamically for which relationships would be difficult to specify by the KS writer in advance.

● BS consists of three main components viz., Knowledge Sources, Blackboard and Control Shell. – BS does not allow direct interaction among modules, as all communication is done via the blackboard through

5.7.Blackboard System Architecture



control shell.

Knowledge Source – KS

- KS can be widely diverse in their internal representation and computational techniques and they do not interact directly with each other.
- KS is a specialist at solving certain aspects of the overall application and is separate and independent of all other KSs.
- Once it finds the information it needs on the blackboard, it can proceed without any assistance from other KSs.
- Additional KSs can be added to the blackboard system, existing KS can be upgraded or even can be removed.
- Each KS is aware of its conditions under which it can contribute toward solving the problem.

Blackboard

- The blackboard is a global data repository and shared data structure available to all KSs
- It contains raw input data, partial solutions and final solutions, control information, communication medium etc.
- The system can retain the results of problem-solved earlier, thus avoiding re-computing them later.
- Structuring of information on blackboard is important issue.
- It should enable a KS to efficiently inspect the blackboard to see if relevant information is present.

Control Shell

- The control shell directs the problem-solving process by allowing KSs to respond opportunistically to changes made to the blackboard.
- The control shell reports about the kind of events in which each KS is interested in.
- It maintains this triggering information and directly considers the KS for activation whenever that kind of event occurs.

Information Representation on Blackboard

● There are two ways for storing information on blackboard viz., – specialized representation and – fully general representation.

● In specialized representation, – KSs may only operate on a few classes of blackboard objects. – Sharing data by only a few KSs limits the extendibility and scalability of the system.

In fully general representation, all aspects of blackboard data are understood by all KSs.

There is a trade-off between these two representations Blackboard System verses Rule Based System ● A blackboard system is different from a rule-based system especially in the size and scope of rules versus the size and complexity of KSs.

● The KSs are substantially larger and more complex than each isomorphic rule in an expert system.

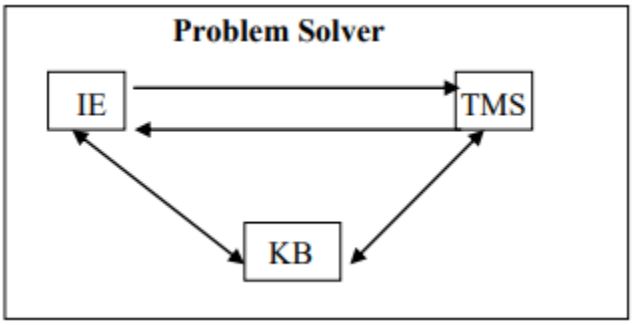
● While expert systems work by firing a rule in response to stimuli, a blackboard system works by executing an entire KS in response to an event.

● A single KS in a blackboard system could be implemented as a complete rule-based system.

Truth Maintenance System (TMS)

● Truth maintenance system (TMS) works with inference engines for solving problems within large search spaces.

● The TMS and inference engine both put together can solve problems where algorithmic solutions do not exist. TMS maintains the beliefs for general problem solving systems.



TMS can be used to implement monotonic or non-monotonic systems.

● In monotonic system, once a fact or piece of knowledge is stored in KB, it can not change.

– In monotonic reasoning, the world of axioms continually increases in size and keeps on expending. – Predicate logic is an example of monotonic form of reasoning. It is a deductive reasoning system where new facts are derived from the known facts

. ● Non-monotonic system allows retraction of truths that are present in the system whenever contradictions arise. – So number of axioms can both increase and decrease and depending upon the changes in KB, it can be updated. Example – Monotonic TMS

● Suppose we are given the premise set $\sum = \{P, W\}$ and the internal constraint set

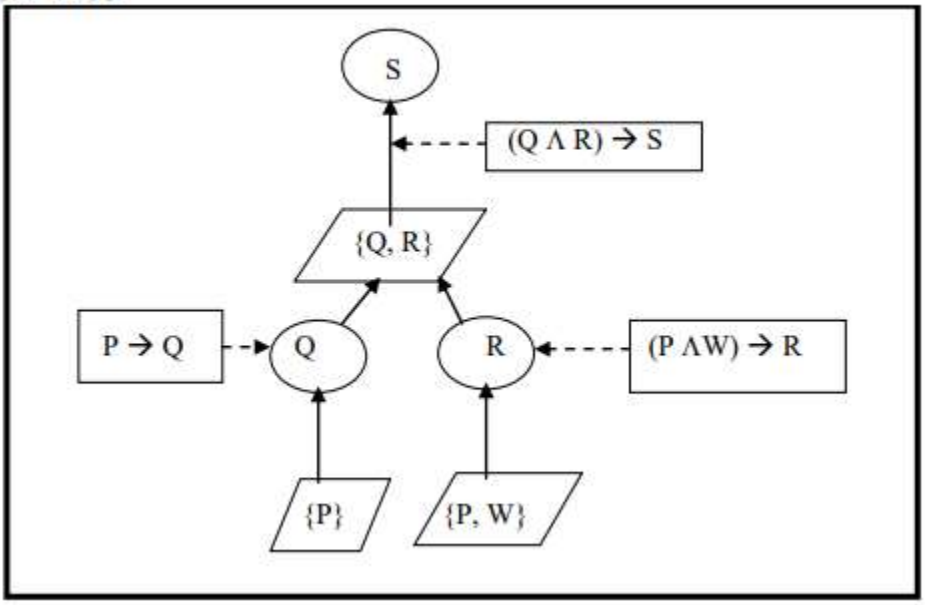
$$\{P \rightarrow Q, (P \wedge W) \rightarrow R, \ (Q \wedge R) \rightarrow S\}.$$

TMS are able to derive S from these constraints and the premise set $\sum$.

● TMS should provide the justifications of deriving S from constraints and premises.

● Therefore, for any given set of internal constraints and premise set $\sum$, if a formula S can be derived from these, then justification functions generate a justification tree for S.

Justification Tree



Non-Monotonic TMS:

● TMS basically operates with two kinds of objects – 'Propositions' declaring different beliefs and – 'Justifications' related to individual propositions for backing up the belief or disbelief expressed by the proposition.

● For every TMS, there are two kinds of justifications required namely 'Support list' and 'Conditional proof'. Support list (SL):

● It is defined as "SL(IN-node)(OUT-node)", where IN-node is a list of all INnodes (propositions) that support the considered node as true. – Here IN means that the belief is true. – OUT-node is a list of all OUT nodes for the considered node to be true. OUT means that belief is not true.

Node number	Facts/assertions	Justification (justified belief)
1	It is sunny	SL(3) (2,4)
2	It rains	SL() ()
3	It is warm	SL(1) (2)
4	It is night time	SL() (1)

CHAPTER III: PROBABILITY THEORY

Topics covered:

joint probability, conditional probability, Bayes's theorem, probabilities in rules and facts of rule based system, cumulative probabilities, rule based system and Bayesian method

---

What Is Joint Probability?

The term joint probability refers to a statistical measure that calculates the likelihood of two events occurring together and at the same point in time. Put simply, a joint probability is the probability of event Y occurring at the same time that event X occurs. In order for joint probability to work, both events must be independent of one another, which means they aren't conditional or don't rely on each other. Joint probabilities can be visualized using Venn diagrams.

- A joint probability is a statistical measure used to calculate the likelihood of two events taking place at the same time.
- Both events must be independent of each other.
- Joint probability is also called the intersection of two or more events.
- It is different from conditional probability, which refers to the probability that one event will happen when another event takes place.
- You can visualize joint probabilities using Venn diagrams.

Formula and Calculation of Joint Probability

Notation for joint probability can take a few different forms. The following formula represents the probability of events intersection:

P (A∩B)**where:**A,B=Two different events that intersectP(A and B),P(AB)=The joint probability of A and B

P (*X*∩*Y*)**where:***X*,*Y*=Two different events that intersect*P*(*X* and *Y*),*P*(*XY*)=The joint probability of X and Y

Although joint probability can help you determine the likelihood of two different events happening at the same time, it does not indicate how the two events may influence each other.

What Does Joint Probability Tell You?

Probability is a field closely related to statistics that deals with the likelihood of an event or phenomenon occurring. It is quantified as a number between 0 and 1, where 0 indicates an impossible chance of occurrence and 1 denotes the certain outcome of an event.

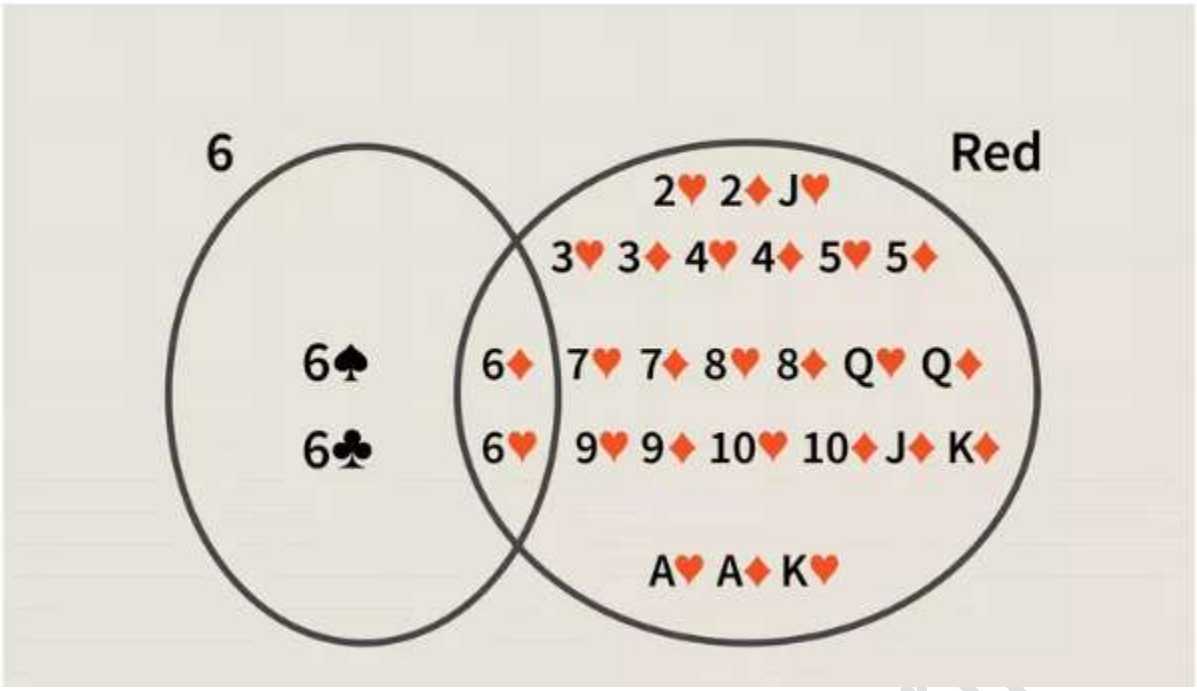
For example, the probability of drawing a red card from a deck of cards is 1/2 = 0.5. This means there is an equal chance of drawing a red and black card since there are 26 of each in a deck. As such, there is a 50-50 probability of drawing a red card versus a black card.

Joint probability measures two events that happen at the same time. It can only be applied to situations where more than one observation can occur at the same time. So the joint probability of picking a card that is both red and 6 from a deck is P(6 ∩ red) = 2/52 = 1/26 since a deck of cards has two red sixes—the six of hearts and the six of diamonds. Because the events red and 6 are independent, you can also use the following formula to calculate the joint probability:

P(6∩red)=P(6)×P(red)=4/52×26/52=1/26*P*(6∩*red*)=*P*(6)×*P*(*red*)=4/52×26/52=1/26

The symbol "∩" in a joint probability is referred to as an intersection. The probability of event X and event Y happening is the same thing as the point where X and Y intersect. Therefore, the joint probability is also called the intersection of two or more events. A Venn diagram is perhaps the best visual tool to explain an intersection:

From the Venn above, the point where both circles overlap is the intersection, which has two observations: the six of hearts and the six of diamonds.



Joint Probability vs. Conditional Probability

Joint probability should not be confused with conditional probability, which is the probability that one event will happen given that another action or event happens. The conditional probability formula is as follows:

$P(A, \text{given } B)$ or $P(A|B)$ or $P(X|Y)$

This is to say that the chance of one event happening is conditional on another event happening. For example, from a deck of cards, the probability that you get a six, given that you drew a red card is $P(6 \mid \text{red}) = 2/26 = 1/13$, since there are two sixes out of 26 red cards.

Joint probability only factors in the likelihood of both events occurring. Conditional probability can be used to calculate joint probability, as seen in this formula:

$P(A \cap B) = P(A|B) \times P(B)$ or $P(X \cap Y) = P(X|Y) \times P(Y)$

The probability that A and B occurs is the probability of X occurring, given that Y occurs multiplied by the probability that Y occurs. Given this formula, the probability of drawing a 6 and a red at the same time will be as follows:

$P(6 \cap \text{red}) = P(6|\text{red}) \times P(\text{red}) = 1/13 \times 26/52 = 1/13 \times 1/2 = 1/26$

$P(6 \cap red) = P(6|red) \times P(red) = 1/13 \times 26/52 = 1/13 \times 1/2 = 1/26$

Statisticians and analysts use joint probability as a tool when two or more observable events can occur simultaneously. For instance, joint probability can be used to estimate the likelihood of a drop in the Dow Jones Industrial Average (DJIA) accompanied by a drop in Microsoft's share price, or the chance that the value of oil rises at the same time the U.S. dollar weakens.

Joint probability depends on the two events acting independently from one another. To determine whether they are truly independent, it's important to establish whether one's outcome affects the other. If they do, they are dependent, which means they lead to conditional probability. If they don't, you end up with joint probability.

Example of Joint Probability

Let's highlight another example to show how joint probability works. This example uses dice and we want to find out what the probability is that you'll roll a four on each die when you roll them. Remember, there are six sides to each one.

In order to determine the joint probability, we first need to determine the probability of each roll:

- The chance of rolling a three on the first die is 1/6
- The chance of rolling a three on the second die is 1/6

Now we can use the joint probability formula noted above to figure out what the joint probability is for this event by multiplying each individual event together.

1/6 x 1/6 = 1/36

This means that there is a 1/36 chance of rolling two fours using a pair of dice.

What Does Joint Probability Tell You?

Joint probability is a statistical measure that tells you the likelihood of two events taking place at the same time. You can use it to determine

What Are the Conditions for Joint Proability?

Certain conditions must be met for joint probability to occur. The first condition is that the two events in question must occur at the same time. Another condition is that both events must occur independently of one another. As such, the outcomes cannot impact each other.

Can Joint Probability Be Greater Than 1?

No, joint probability can never be greater than 1. Joint probability falls between 0 and 1, where 0 denotes that the likelihood of two events occurring simultaneously is impossible while 1 indicates that their outcome is certain.

The Bottom Line

Probability refers to the likelihood that an event will take place. But when there are two variables involved, you may end up with joint probability. This is a statistical measure that can tell you whether two independent events are likely to occur at the same time. It is an important metric for statisticians who use it to determine relationships between two sets of variables, such as women and sports. But one thing it doesn't indicate, though, is how the two influence each other.

Uncertainty

- Knowledge representation with first-order logic and propositional logic is based on certainty, means we are sure about the predicates. For example, we can say that A→B, which means if A is true then B is true, but what about the situation where we are not sure about whether A is true or not then we cannot express this statement, this situation comes under uncertainty.
- So to characterize uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

Causes of uncertainty:

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

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

Need of Probabilistic Reasoning in AI

- **Unpredictable outcomes**
- **Predicates are too large to handle**
- **Unknown error occurs**

In probabilistic reasoning, there are two methods to solve difficulties with uncertain knowledge:

- **Bayes' rule**
- **Bayesian Statistics**
- **Probability:** Probability can be defined as chance of occurrence of an uncertain event. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1.

$0 \leq P(X) \leq 1$, where P(X) is the probability of an event X.

- P(X) = 0, indicates total uncertainty in an event X.
- P(X) =1, indicates total certainty in an event X.

Bayes' Theorem in Artificial Intelligence

Bayes' theorem determines the probability of an event with uncertain knowledge.

It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).

- **Example**: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.

Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:

As from product rule we can write: **$P(A \wedge B) = P(A|B)\,P(B)$**

Similarly, the probability of event B with known event A: **$P(A \wedge B) = P(B|A)\,P(A)$**, here $P(A \wedge B)$ is joint probability.

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

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

- The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.
- Here, P(A|B) is known as **posterior**, which we need to calculate and it will be read as Probability of hypothesis A with occurrence of an evidence B.
- P(B|A) - likelihood, in which we consider that hypothesis is true, then we compute the likelihood of evidence.
- P(A) - **prior probability**, probability of hypothesis before seeing the evidence
- P(B) - **marginal probability** of an evidence.
- In the equation (a), in general, we can write P (B) = P(A)*P(B|Ai), hence the Bayes' rule can be written as:

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

- Where $A_1, A_2, \ldots\ldots, A_n$ is a set of mutually exclusive and exhaustive events.

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

•Solution:

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

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

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

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

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

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

Bayesian Belief Network

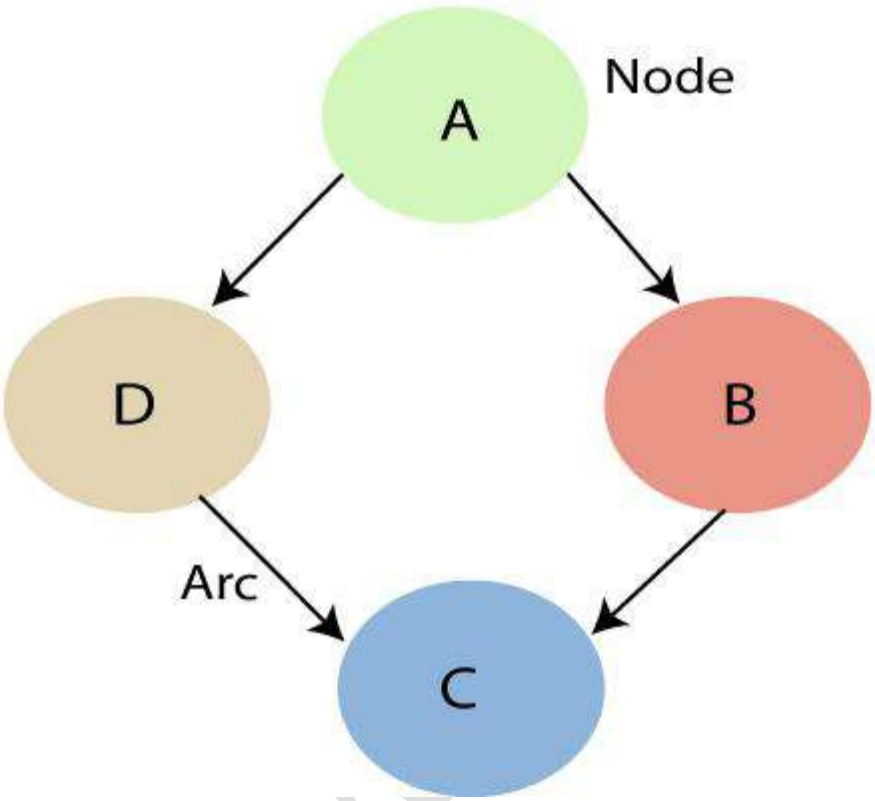
Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- **Directed Acyclic Graph**
- **Table of conditional probabilities**

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.

Note: It is used to represent conditional dependencies.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.

- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph. These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
- **Note: The Bayesian network graph does not contain any cyclic graph. Hence, it is known as a directed acyclic graph or DAG.**
- The Bayesian network has mainly two components:

1.Causal Component

2.Actual numbers

- Each node in the Bayesian network has condition probability distribution **P(Xi |Parent(Xi))**, which determines the effect of the parent on that node.
- Bayesian network is based on Joint probability distribution and conditional probability.

Example: Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and

Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

•Problem: Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

Note: List of all events occurring in this network:

Burglary (B)
Earthquake(E)
Alarm(A)
David Calls(D)
Sophia calls(S)

•From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E)

- **Example:** Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

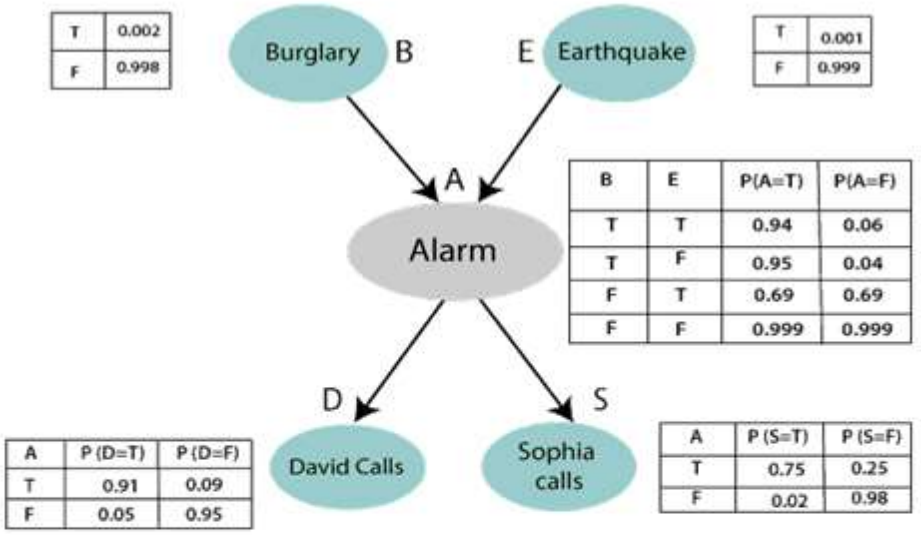
- **Problem: Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.**

Note: List of all events occurring in this network:

Burglary (B)
Earthquake(E)
Alarm(A)
David Calls(D)
Sophia calls(S)

- From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E)



T	0.002
F	0.998

T	0.001
F	0.999

B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P (D=T)	P (D=F)
T	0.91	0.09
F	0.05	0.95

A	P (S=T)	P (S=F)
T	0.75	0.25
F	0.02	0.98

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E).
= 0.75* 0.91* 0.001* 0.998*0.999
= 0.00068045.
Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

Cumulative Probability Distribution

Cumulative Probability Distribution takes value in a continuous range; for example, the range may consist of a set of real numbers. In this case, Cumulative Probability Distribution will take any value from the continuum of real numbers unlike the discrete or some finite value taken in the case of Discrete Probability distribution. Cumulative Probability Distribution is of two types, **Continuous Uniform Distribution, and Normal Distribution.**

Continuous Uniform Distribution

Continuous Uniform Distribution is described by a density function that is flat and assumes value in a closed interval let's say [P, Q] such that the probability is uniform in this closed interval. It is represented as f(x; P, Q)

f(x; P, Q) = 1/(Q-P) for P≤x≤Q
f(x; P, Q) = 0; elsewhere

Normal Distribution

Normal Distribution of continuous random variables results in a bell-shaped curve. It is often referred to as Gaussian Distribution on the name of Karl Friedrich Gauss who derived its equation. This curve is frequently used by the meteorological department for rainfall studies. The Normal Distribution of random variable X is given by

n(x; μ, σ) = {1/(√2π)σ}e(-1/2σ^2)(x-μ)^2 for -∞<x<∞
where

- ***μ*** *is mean*
- ***σ*** *is variance*

Normal Distribution Examples

The Normal Distribution Curve can be used to show the distribution of natural events very well. Over the period it has become a favorite choice of statisticians to study natural events. Some of the examples where the Normal Distribution Curve can be used are mentioned below

- Salary of Working Class
- Life Expectancy of human in a Country
- Heights of Male or Female
- The IQ Level of children
- Expenditure of households

Probability Distribution Function

Probability Distribution Function is defined as the function that is used to express the distribution of a probability. Different types of probability, they are expressed differently. These functions are also used for Probability Density Functions for different variables.

For Normal Distribution, the Probability Distribution Function for Random Variable X is given by **Fx(x) = P(X ≤ x)** where X is the Random variable and P is the Probability.
The Cumulative Probability Distribution for closed interval a⇢b is given as **P(a < X ≤ b) = Fx(b) – Fx(a)**.

In terms of integrals, the cumulative probability function is given as
For Random Variable X = p, the Cumulative Probability function is given

as
Binomial Probability Distribution gives some exact values. It is often called as Probability Mass Function. For a Random Variable X and Space S where X: S⇢A where A belongs to Random Discrete Variable R, X can be defined as **fx(x) = Pr(X = x) = P({s ∈ S: X(s) = x}).**

Probability Distribution Table

The random variables and their corresponding probability is tabulated then it is called Probability Distribution Table. The following table represents a Probability Distribution Table

X	X1	X2	X3	X4	….	Xn
P(X)	P1	P2	P3	P4	….	Pn

It should be noted that the sum of all probabilities is equal to 1.

Prior Probability

Prior Probability as the name suggests refers to assigning the probability of an event before the happening of a dependent event that makes us make changes in the Prior Probability. Let's say we assign Probability P(A) to event A before taking into account that event B has also happened. After B has happened we need to revise P(A) using Baye's Theorem. Hence, here P(A) is the Prior Probability. If we predict that a particular observation will fall into a particular category before collecting all the observations, then this is also called Prior Probability.

Posterior Probability

After the Prior Probability has been assigned and new information is obtained then the Prior Probability is modified by taking into account the newly obtained information using Baye's Formula. This revised probability is called Posterior Probability. Hence, we can say that Posterior Probability is a conditional probability obtained by revising the Prior Probability.

Solved Examples on Probability Distribution

Example 1: A box contains 4 blue balls and 3 green balls. Find the probability distribution of the number of green balls in a random draw of 3 balls.

Solution:

Given that the total number of balls is 7 out of which 3 have to be drawn at random. On drawing 3 balls the possibilities are all 3 are green, only 2 is green, only 1 is green, and no green. Hence X = 0, 1, 2, 3.
P(No ball is green) = P(X = 0) = 4C3/7C3 = 4/35
P(1 ball is green) = P(X = 1) = 3C1 × 4C2/7C3 = 18/35
P(2 balls are green) = P(X = 2) = 3C2 × 4C1/7C3 = 12/35
P(All 3 balls are green) = P(X = 3) = 3C3/7C3 = 1/35
Hence, the probability distribution for this problem is given as follows

X	0	1	2	3
P(X)	4/35	18/35	12/35	1/35

Rule-Based System

What is a rule-based system in AI?

A rule-based system is a system that applies human-made rules to store, sort and manipulate data. In doing so, it mimics human intelligence.
Rule-based systems require a set of facts or source of data, and a set of rules for manipulating that data. These rules are sometimes referred to as 'If statements' as they tend to follow the line of 'IF X happens THEN do Y'.
The steps can be simplified to:

- First comes the data or new business event
- Then comes the analysis: the part where the system conditionally processes the data against its rules
- Then comes any subsequent automated follow-up actions

Some of the important elements of rule-based systems include:

A set of facts

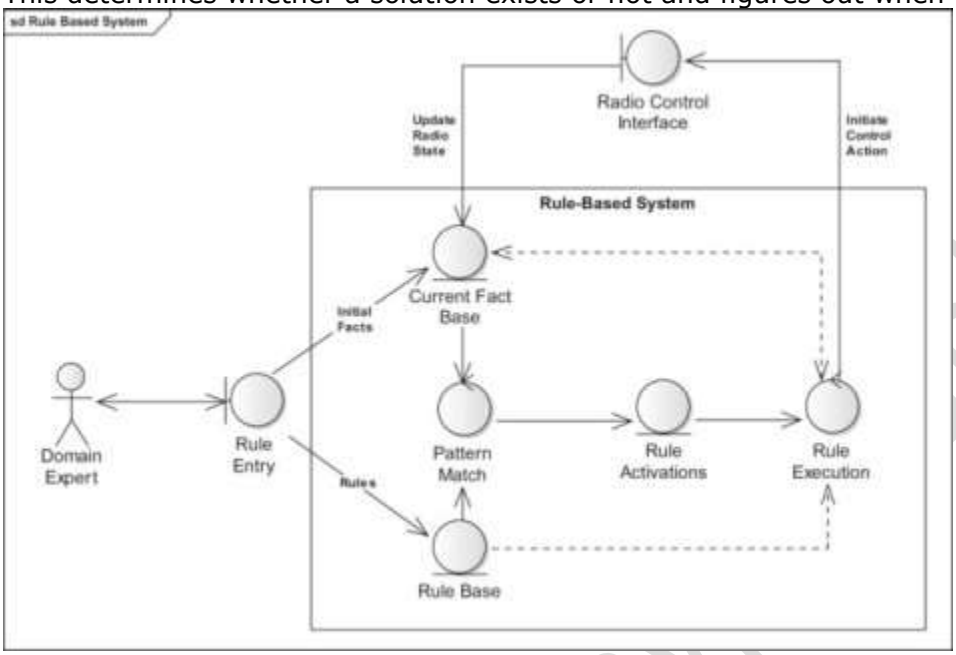
These facts are assertions or anything that is relevant to the beginning state of the system.

Set of Rules

This set contains all the actions that should be performed within the scope of a problem and defines how to act on the assertion set. In the set of rules facts are represented in an IF-THEN form.

Termination Criteria or Interpreter

This determines whether a solution exists or not and figures out when the process should be terminated.



Source: Science Direct

What are the characteristics of rule-based systems?

Some of the features of rule-based systems are:

- They are made up of the combined knowledge of human experts in the problem domain.
- They represent knowledge in a very declarative manner.
- They make it possible to use various knowledge representations paradigms.
- They support the implementation of non-deterministic search and control strategies.
- They help in describing fragmentary, ill-structured, heuristic, judgemental knowledge.
- They are robust and have the ability to operate using uncertain or incomplete knowledge.
- They can help with rule based decision making.

How does a rule based system in AI work?

Rule-based systems outlines triggers & the actions that should follow (or are triggered). For example, a trigger might be an email containing the word "invoice". An action might then be to forward the email to the finance team.

These rules most often take the form of "if" statements. "IF" outlines the trigger, "THEN" specifies the action to complete. So, if you want to create a rule-based system capable of handling 100 different actions, you'd have to write 100 different rules. If you want to then update the system and add actions, then you would need to write new rules.

In short, you use rules to tell a machine what to do, and the machine will do exactly as you tell it. From there, rule-based systems will execute the actions until you tell it to stop.

But remember: if you tell it to do something incorrectly, it will do it incorrectly.

What are the main components of a rules-based system?

A typical rule-based system has seven basic components:

The knowledge base

It holds the domain knowledge that is necessary for problem solving. In a rules-based system, the knowledge gets represented as a set of rules. Every rule specifies a relation, recommendation, directive, strategy or heuristic and has the IF (condition) THEN (action) structure. As soon as the condition part of the rule is satisfied, the rule gets triggered and the action part gets executed.

The database

The database has a set of facts that are used to compare against the IF (condition) part of the rules that are held in the knowledge base.

The inference engine

The inference engine is used to perform the reasoning through which the expert system comes to a solution. The job of the inference engine is to link the rules that are defined in the knowledge base with the facts that are stored in the database. The inference engine is also known as the semantic reasoner. It infers information or performs required actions on the basis of input and the rule base that's present in the knowledge base. The semantic reasoner involves a match-resolve-act cycle that works like this:

- Match - A section of the production rule system gets matched with the contents of the working memory to gain a conflict, where there are several instances of the satisfied productions.
- Conflict-Resolution - After the production system is matched, one of the production instances in the conflict is selected for execution for the purpose of determining the progress of the process.
- Act - The production instance selected in the previous stage is s executed, impacting the contents of the working memory.

Explanation facilities

The explanantion facilities make it possible for the user to ask the expert system how a specific conclusion was reached and why a specific fact is required. The expert system needs to be able to explain its reasoning and justify its advice, analysis, or conclusion.

User interface

The user interface is the means through which the user seeking a solution to a problem communicates with the expert system. The communication should be as meaningful and friendly as possible and the user interface should be as intuitive as possible.

These five elements are critical for any rule-based system. They are the core components of the rule-based system. But the system might have some additional components as well. A couple of these components could be the external interface and the working memory.

External interface

The external interface enables an expert system to work with external data files and programs that are written in conventional programming languages like C, Pascal, FORTRAN and Basic.

Working memory

The working memory stores temporary information and data.

How are rules-based systems different from learning-based systems?

In contrast to Rule-Based Systems, Learning Systems observe data and continuously learn from it.

That is, while Rules Based Systems quickly become out of date, Learning Systems automatically improve over time. And Learning Systems improve without the massive expense of having people maintain complex rules.
Learning systems find patterns and treat similar things similarly. Think of how Netflix or Hulu keep track and learn from what you watch, and then compare it to what people like you watch to make recommendations and keep you binge-watching. There is some complexity in execution, but it's a much simpler concept. And a concept more in line with how humans also learn.
Finally, Learning Systems' results are necessarily more measurable. It's the only way they can improve over time; because if they cannot measure results, they could not learn what actions are better and what actions are worse.

Construction of rule-based systems
Data-based construction follows a machine learning method, which is in general domain-independent. Machine learning techniques can be categorized into two sub-types: supervised learning and unsupervised learning.
Supervised learning means learning with assistance. This is because all instances from a data set are labeled. This type of learning aims to predict attribute values for unknown instances by using the known data instances. The predicted value of an attribute may be either discrete or continuous in the construction. Therefore, supervised learning could be involved in both classification and regression tasks that can be used for categorical prediction and numerical prediction respectively.
On the other hand, unsupervised learning means learning without any assistance. This is because all instances from a data set are unlabeled. This type of learning aims to find earlier unknown patterns from data sets. It includes association, which aims to find relationships among attributes about their values, and clustering, which aims to find a group of objects that are similar from data sets.
In general, all three types of rule-based systems can be constructed with the following steps: Data collection->Data pre-processing->Learning from data->Testing. However, there are different requirements for different learning tasks.
In other words, to build a high-quality model by using machine learning techniques, it is important to find algorithms that are suitable for the chosen data sets concerning the characteristics of data. From this point of view, data preprocessing may not be necessary if the chosen algorithms are good fits. In addition, different types of dimensionality reduction techniques (such as feature selection), a type of data preprocessing, may be required for different tasks. If it is a classification or regression task, supervised feature selection techniques may be required in general.

What are advantages of the rule-based system in AI?
1. A rule-based system is generally cost-efficient and accurate in terms of its results.
2. The outputs generated by the system are dependent on rules so the output responses are stable and not random.
3. The coverage for different circumstances is less, whatever scenarios are covered by the Rule Based system will provide high accuracy. The error rate goes down because of the predefined rules.
4. It's feasible to reduce the amount of risk in terms of system accuracy.
5. Optimizing the speed of the system is easier as you know all the parts. So providing instant outputs, is not a big issue.

What are the disadvantages of the rule-based system in AI?

1. A rule-based system is built upon a lot of data, deep knowledge of the domain, and a lot of manual work.
2. Writing and generating rules for a complex system is quite challenging and time-consuming.
3. The self-learning capacity in a rule-based system is less as it generates the result as per the rules.
4. Complex pattern identification is a challenging task in the Rule Based method as it takes a lot of time and analysis.

CHAPTER IV: Fuzzy Sets and Fuzzy Logic

Topics covered:

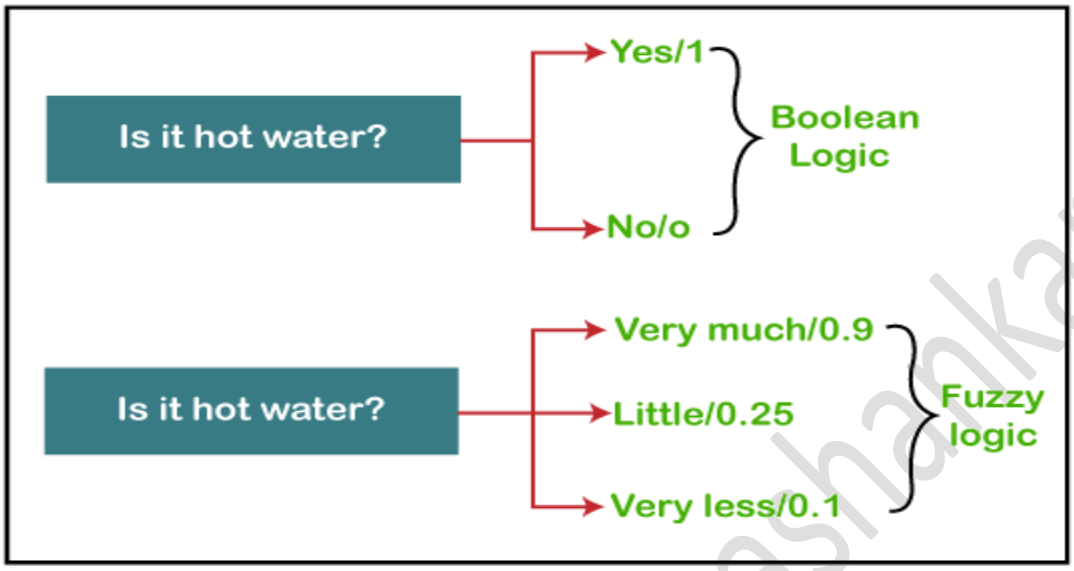
Fuzzy Sets, Fuzzy set operations, Types of Member ship Functions, Multivalued Logic, Fuzzy Logic, Linguistic variables and Hedges, Fuzzy propositions, inference rules for fuzzy propositions, fuzzy systems, possibility theory and other enhancement to Logic

---

What is Fuzzy Logic?

The **'Fuzzy'** word means the things that are not clear or are vague. Sometimes, we cannot decide in real life that the given problem or statement is either true or false. At that time, this concept provides many values between the true and false and gives the flexibility to find the best solution to that problem.

Example of Fuzzy Logic as comparing to Boolean Logic



Fuzzy logic contains the multiple logical values and these values are the truth values of a variable or problem between 0 and 1. This concept was introduced by **Lofti Zadeh** in **1965** based on the **Fuzzy Set Theory**. This concept provides the possibilities which are not given by computers, but similar to the range of possibilities generated by humans.

In the Boolean system, only two possibilities (0 and 1) exist, where 1 denotes the absolute truth value and 0 denotes the absolute false value. But in the fuzzy system, there are multiple possibilities present between the 0 and 1, which are partially false and partially true.

The Fuzzy logic can be implemented in systems such as micro-controllers, workstation-based or large network-based systems for achieving the definite output. It can also be implemented in both hardware or software.

Characteristics of Fuzzy Logic

Following are the characteristics of fuzzy logic:

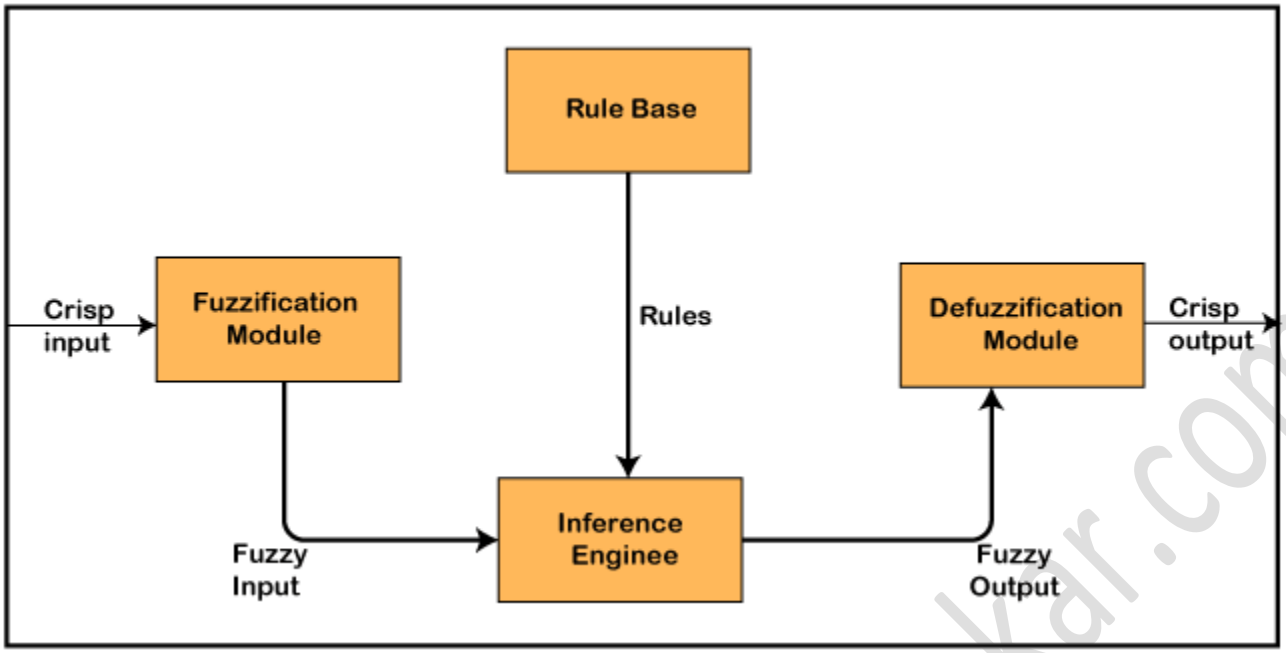
1. This concept is flexible and we can easily understand and implement it.
2. It is used for helping the minimization of the logics created by the human.
3. It is the best method for finding the solution of those problems which are suitable for approximate or uncertain reasoning.
4. It always offers two values, which denote the two possible solutions for a problem and statement.
5. It allows users to build or create the functions which are non-linear of arbitrary complexity.
6. In fuzzy logic, everything is a matter of degree.
7. In the Fuzzy logic, any system which is logical can be easily fuzzified.
8. It is based on natural language processing.
9. It is also used by the quantitative analysts for improving their algorithm's execution.
10. It also allows users to integrate with the programming.

Architecture of a Fuzzy Logic System

In the architecture of the **Fuzzy Logic** system, each component plays an important role. The architecture consists of the different four components which are given below.

1. Rule Base
2. Fuzzification
3. Inference Engine
4. Defuzzification

Following diagram shows the architecture or process of a Fuzzy Logic system:



1. Rule Base

Rule Base is a component used for storing the set of rules and the If-Then conditions given by the experts are used for controlling the decision-making systems. There are so many updates that come in the Fuzzy theory recently, which offers effective methods for designing and tuning of fuzzy controllers. These updates or developments decreases the number of fuzzy set of rules.

2. Fuzzification

Fuzzification is a module or component for transforming the system inputs, i.e., it converts the crisp number into fuzzy steps. The crisp numbers are those inputs which are measured by the sensors and then fuzzification passed them into the control systems for further processing. This component divides the input signals into following five states in any Fuzzy Logic system:

- Large Positive (LP)
- Medium Positive (MP)
- Small (S)
- Medium Negative (MN)
- Large negative (LN)

3. Inference Engine

This component is a main component in any Fuzzy Logic system (FLS), because all the information is processed in the Inference Engine. It allows users to find the matching degree between the current fuzzy input and the rules. After the matching degree, this system determines which rule is to be added according to the given input field. When all rules are fired, then they are combined for developing the control actions.

4. Defuzzification

Defuzzification is a module or component, which takes the fuzzy set inputs generated by the **Inference Engine**, and then transforms them into a crisp value. It is the last step in the process of a fuzzy logic system. The crisp value is a type of value which is acceptable by the user. Various techniques are present to do this, but the user has to select the best one for reducing the errors.

Membership Function

The membership function is a function which represents the graph of fuzzy sets, and allows users to quantify the linguistic term. It is a graph which is used for mapping each element of x to the value between 0 and 1.

This function is also known as indicator or characteristics function.

This function of Membership was introduced in the first papers of fuzzy set by **Zadeh**. For the Fuzzy set B, the membership function for X is defined as: $\mu B:X \rightarrow [0,1]$. In this function X, each element of set B is mapped to the value between 0 and 1. This is called a degree of membership or membership value.

Classical and Fuzzy Set Theory

To learn about classical and Fuzzy set theory, firstly you have to know about what is set.

Set

A set is a term, which is a collection of unordered or ordered elements. Following are the various examples of a set:

1. A set of all-natural numbers
2. A set of students in a class.
3. A set of all cities in a state.
4. A set of upper-case letters of the alphabet.

Types of Set:

There are following various categories of set:

1. Finite
2. Empty
3. Infinite
4. Proper
5. Universal
6. Subset
7. Singleton
8. Equivalent Set
9. Disjoint Set

Classical Set

It is a type of set which collects the distinct objects in a group. The sets with the crisp boundaries are classical sets. In any set, each single entity is called an element or member of that set.

Mathematical Representation of Sets

Any set can be easily denoted in the following two different ways:

1. Roaster Form: This is also called as a tabular form. In this form, the set is represented in the following way:

Set_name = { element1, element2, element3, ..., element N}

The elements in the set are enclosed within the brackets and separated by the commas.

Following are the two examples which describes the set in Roaster or Tabular form:

Example 1:

Set of Natural Numbers: N={1, 2, 3, 4, 5, 6, 7, ...,n).

Example 2:

Set of Prime Numbers less than 50: X={2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47}.

2. Set Builder Form: Set Builder form defines a set with the common properties of an element in a set. In this form, the set is represented in the following way:

A = {x:p(x)}

The following example describes the set in the builder form:

The set {2, 4, 6, 8, 10, 12, 14, 16, 18} is written as:

B = {x:2 ≤ x < 20 and (x%2) = 0}

Operations on Classical Set

Following are the various operations which are performed on the classical sets:

1. Union Operation
2. Intersection Operation
3. Difference Operation
4. Complement Operation

1. Union:

This operation is denoted by (A U B). A U B is the set of those elements which exist in two different sets A and B. This operation combines all the elements from both the sets and make a new set. It is also called a Logical OR operation.

It can be described as:

A ∪ B = { x | x ∈ A OR x ∈ B }.

Example:

Set A = {10, 11, 12, 13}, Set B = {11, 12, 13, 14, 15}, then A ∪ B = {10, 11, 12, 13, 14, 15}

2. Intersection

This operation is denoted by (A ∩ B). A ∩ B is the set of those elements which are common in both set A and B. It is also called a Logical OR operation.

It can be described as:

A ∩ B = { x | x ∈ A AND x ∈ B }.

Example:

Set A = {10, 11, 12, 13}, Set B = {11, 12, 14} then A ∩ B = {11, 12}

3. Difference Operation

This operation is denoted by (A - B). A-B is the set of only those elements which exist only in set A but not in set B.
It can be described as:
A - B = { x | x ∈ A AND x ∉ B }.
4. Complement Operation: This operation is denoted by (A`). It is applied on a single set. A` is the set of elements which do not exist in set A.
It can be described as:
A′ = {x|x ∉ A}.
Properties of Classical Set
There are following various properties which play an essential role for finding the solution of a fuzzy logic problem.
1. Commutative Property:
This property provides the following two states which are obtained by two finite sets A and B:
A ∪ B = B ∪ A
A ∩ B = B ∩ A
2. Associative Property:
This property also provides the following two states but these are obtained by three different finite sets A, B, and C:
A ∪ (B ∪ C) = (A ∪ B) ∪ C
A ∩ (B ∩ C) = (A ∩ B) ∩ C
3. Idempotency Property:
This property also provides the following two states but for a single finite set A:
A ∪ A = A
A ∩ A = A
4. Absorption Property
This property also provides the following two states for any two finite sets A and B:
A ∪ (A ∩ B) = A
A ∩ (A ∪ B) = A
5. Distributive Property:
This property also provides the following two states for any three finite sets A, B, and C:
A∪ (B ∩ C) = (A ∪ B)∩ (A ∪ C)
A∩ (B ∪ C) = (A∩B) ∪ (A∩C)
6. Identity Property:
This property provides the following four states for any finite set A and Universal set X:
A ∪ φ =A
A ∩ X = A
A ∩ φ = φ
A ∪ X = X
7. Transitive property
This property provides the following state for the finite sets A, B, and C:
If A ⊆ B ⊆ C, then A ⊆ C
8. Ivolution property
This property provides following state for any finite set A:

$$\overline{\overline{A}} = A$$

9. De Morgan's Law
This law gives the following rules for providing the contradiction and tautologies:

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

Fuzzy Set
The set theory of classical is the subset of Fuzzy set theory. Fuzzy logic is based on this theory, which is a generalisation of the classical theory of set (i.e., crisp set) introduced by Zadeh in 1965.
A fuzzy set is a collection of values which exist between 0 and 1. Fuzzy sets are denoted or represented by the tilde (~) character. The sets of Fuzzy theory were introduced in 1965 by Lofti A. Zadeh and Dieter

Klaua. In the fuzzy set, the partial membership also exists. This theory released as an extension of classical set theory.
This theory is denoted mathematically asA fuzzy set (Ã) is a pair of U and M, where U is the Universe of discourse and M is the membership function which takes on values in the interval [0, 1]. The universe of discourse (U) is also denoted by Ω or X.

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) | x \in X\}$$

Operations on Fuzzy Set
Given Ã and B are the two fuzzy sets, and X be the universe of discourse with the following respective member functions:

$\mu_{\tilde{A}}(x)$ and $\mu_{\tilde{B}}(x)$

The operations of Fuzzy set are as follows:
1. Union Operation: The union operation of a fuzzy set is defined by:
$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$
Example:
Let's suppose A is a set which contains following elements:
$A = \{(X_1, 0.6), (X_2, 0.2), (X_3, 1), (X_4, 0.4)\}$
And, B is a set which contains following elements:
$B = \{(X_1, 0.1), (X_2, 0.8), (X_3, 0), (X_4, 0.9)\}$
then,
$A \cup B = \{(X_1, 0.6), (X_2, 0.8), (X_3, 1), (X_4, 0.9)\}$
Because, according to this operation
For X_1
$\mu_{A \cup B}(X_1) = \max(\mu_A(X_1), \mu_B(X_1))$
$\mu_{A \cup B}(X_1) = \max(0.6, 0.1)$
$\mu_{A \cup B}(X_1) = 0.6$
For X_2
$\mu_{A \cup B}(X_2) = \max(\mu_A(X_2), \mu_B(X_2))$
$\mu_{A \cup B}(X_2) = \max(0.2, 0.8)$
$\mu_{A \cup B}(X_2) = 0.8$
For X_3
$\mu_{A \cup B}(X_3) = \max(\mu_A(X_3), \mu_B(X_3))$
$\mu_{A \cup B}(X_3) = \max(1, 0)$
$\mu_{A \cup B}(X_3) = 1$
For X_4
$\mu_{A \cup B}(X_4) = \max(\mu_A(X_4), \mu_B(X_4))$
$\mu_{A \cup B}(X_4) = \max(0.4, 0.9)$
$\mu_{A \cup B}(X_4) = 0.9$
2. Intersection Operation: The intersection operation of fuzzy set is defined by:
$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$
Example:
Let's suppose A is a set which contains following elements:
$A = \{(X_1, 0.3), (X_2, 0.7), (X_3, 0.5), (X_4, 0.1)\}$
And, B is a set which contains following elements:
$B = \{(X_1, 0.8), (X_2, 0.2), (X_3, 0.4), (X_4, 0.9)\}$
then,
$A \cap B = \{(X_1, 0.3), (X_2, 0.2), (X_3, 0.4), (X_4, 0.1)\}$
Because, according to this operation
For X_1
$\mu_{A \cap B}(X_1) = \min(\mu_A(X_1), \mu_B(X_1))$
$\mu_{A \cap B}(X_1) = \min(0.3, 0.8)$
$\mu_{A \cap B}(X_1) = 0.3$
For X_2
$\mu_{A \cap B}(X_2) = \min(\mu_A(X_2), \mu_B(X_2))$
$\mu_{A \cap B}(X_2) = \min(0.7, 0.2)$
$\mu_{A \cap B}(X_2) = 0.2$
For X_3

$\mu_{A\cap B}(X_3) = \min(\mu_A(X_3), \mu_B(X_3))$
$\mu_{A\cap B}(X_3) = \min(0.5, 0.4)$
$\mu_{A\cap B}(X_3) = 0.4$
For X_4
$\mu_{A\cap B}(X_4) = \min(\mu_A(X_4), \mu_B(X_4))$
$\mu_{A\cap B}(X_4) = \min(0.1, 0.9)$
$\mu_{A\cap B}(X_4) = 0.1$
3. Complement Operation: The complement operation of fuzzy set is defined by:
$\mu_{\bar{A}}(x) = 1-\mu_A(x)$,
Example:
Let's suppose A is a set which contains following elements:
$A = \{(X_1, 0.3), (X_2, 0.8), (X_3, 0.5), (X_4, 0.1)\}$
then,
$\bar{A} = \{(X_1, 0.7), (X_2, 0.2), (X_3, 0.5), (X_4, 0.9)\}$
Because, according to this operation
For X_1
$\mu_{\bar{A}}(X_1) = 1-\mu_A(X_1)$
$\mu_{\bar{A}}(X_1) = 1 - 0.3$
$\mu_{\bar{A}}(X_1) = 0.7$
For X_2
$\mu_{\bar{A}}(X_2) = 1-\mu_A(X_2)$
$\mu_{\bar{A}}(X_2) = 1 - 0.8$
$\mu_{\bar{A}}(X_2) = 0.2$
For X_3
$\mu_{\bar{A}}(X_3) = 1-\mu_A(X_3)$
$\mu_{\bar{A}}(X_3) = 1 - 0.5$
$\mu_{\bar{A}}(X_3) = 0.5$
For X_4
$\mu_{\bar{A}}(X_4) = 1-\mu_A(X_4)$
$\mu_{\bar{A}}(X_4) = 1 - 0.1$
$\mu_{\bar{A}}(X_4) = 0.9$

Classical Set Theory	Fuzzy Set Theory
1. This theory is a class of those sets having sharp boundaries.	1. This theory is a class of those sets having un-sharp boundaries.
2. This set theory is defined by exact boundaries only 0 and 1.	2. This set theory is defined by ambiguous boundaries.
3. In this theory, there is no uncertainty about the boundary's location of a set.	3. In this theory, there always exists uncertainty about the boundary's location of a set.
4. This theory is widely used in the design of digital systems.	4. It is mainly used for fuzzy controllers.

Applications of Fuzzy Logic
Following are the different application areas where the Fuzzy Logic concept is widely used:
1. It is used in **Businesses** for decision-making support system.
2. It is used in **Automative systems** for controlling the traffic and speed, and for improving the efficiency of automatic transmissions. **Automative systems** also use the shift scheduling method for automatic transmissions.
3. This concept is also used in the **Defence** in various areas. Defence mainly uses the Fuzzy logic systems for underwater target recognition and the automatic target recognition of thermal infrared images.
4. It is also widely used in the **Pattern Recognition and Classification** in the form of Fuzzy logic-based recognition and handwriting recognition. It is also used in the searching of fuzzy images.
5. Fuzzy logic systems also used in **Securities**.
6. It is also used in **microwave oven** for setting the lunes power and cooking strategy.

7. This technique is also used in the area of **modern control systems** such as expert systems.
8. **Finance** is also another application where this concept is used for predicting the stock market, and for managing the funds.
9. It is also used for controlling the brakes.
10. It is also used in the **industries of chemicals** for controlling the ph, and chemical distillation process.
11. It is also used in the **industries of manufacturing** for the optimization of milk and cheese production.
12. It is also used in the vacuum cleaners, and the timings of washing machines.
13. It is also used in heaters, air conditioners, and humidifiers.

Advantages of Fuzzy Logic

Fuzzy Logic has various advantages or benefits. Some of them are as follows:

1. The methodology of this concept works similarly as the human reasoning.
2. Any user can easily understand the structure of Fuzzy Logic.
3. It does not need a large memory, because the algorithms can be easily described with fewer data.
4. It is widely used in all fields of life and easily provides effective solutions to the problems which have high complexity.
5. This concept is based on the set theory of mathematics, so that's why it is simple.
6. It allows users for controlling the control machines and consumer products.
7. The development time of fuzzy logic is short as compared to conventional methods.
8. Due to its flexibility, any user can easily add and delete rules in the FLS system.

Disadvantages of Fuzzy Logic

Fuzzy Logic has various disadvantages or limitations. Some of them are as follows:

1. The run time of fuzzy logic systems is slow and takes a long time to produce outputs.
2. Users can understand it easily if they are simple.
3. The possibilities produced by the fuzzy logic system are not always accurate.
4. Many researchers give various ways for solving a given statement using this technique which leads to ambiguity.
5. Fuzzy logics are not suitable for those problems that require high accuracy.
6. The systems of a Fuzzy logic need a lot of testing for verification and validation.

Linguistic Variables and Hedges

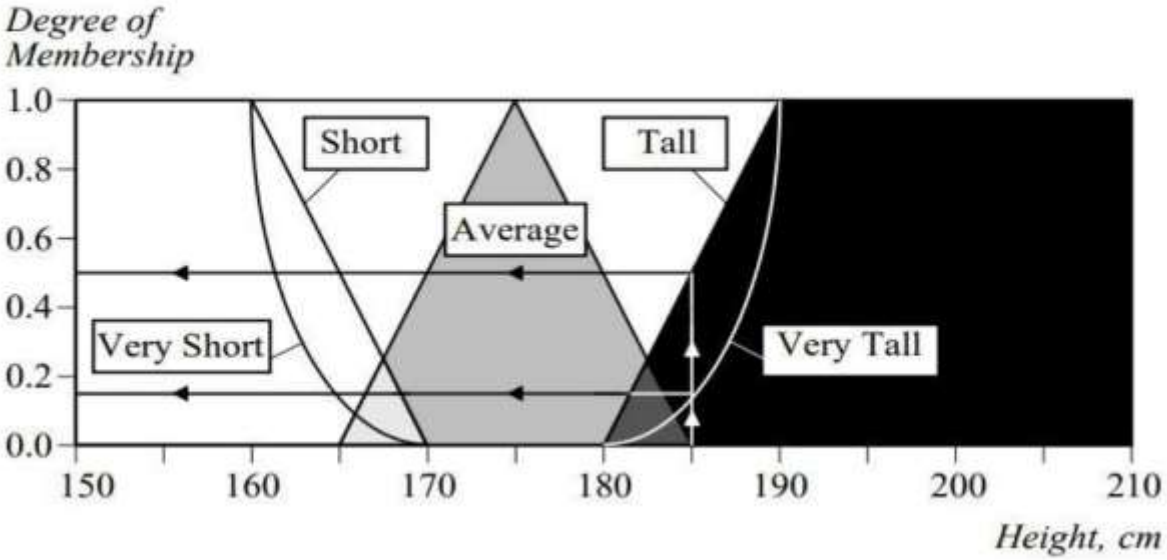
▪ At the root of fuzzy set theory lies the idea of linguistic variables.
▪ A linguistic variable is a fuzzy variable. For example, the statement "John's height is tall" implies that the linguistic variable height takes the linguistic value tall.

In fuzzy expert systems, linguistic variables are used in fuzzy rules.
For example:
IF wind is strong
THEN sailing is good
IF project_duration is long
THEN completion_risk is high
IF speed is slow
THEN stopping_distance is short

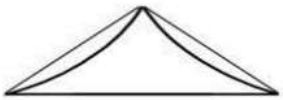
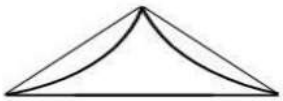
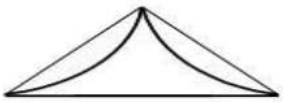
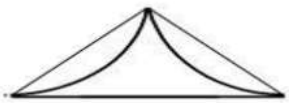
The range of possible values of a linguistic variable represents the universe of discourse of that variable. For example, the universe of discourse of the linguistic variable speed might have the range between 0 and 220 km/h and may include such fuzzy subsets as very slow, slow, medium, fast and very fast.

▪ A linguistic variable carries with it the concept of fuzzy set qualifiers, called hedges.
▪ Hedges are terms that modify the shape of fuzzy sets. They include adverbs such as very, somewhat, quite, more or less and slightly

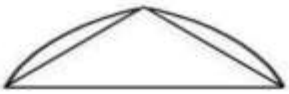
Fuzzy sets with the hedge very



Representation of hedges in fuzzy logic

Hedge	*Mathematical Expression*	*Graphical Representation*
A little	$[\mu_A(x)]^{1.3}$	
Slightly	$[\mu_A(x)]^{1.7}$	
Very	$[\mu_A(x)]^{2}$	
Extremely	$[\mu_A(x)]^{3}$	

Representation of hedges in fuzzy logic (continued)

Hedge	*Mathematical Expression*	*Graphical Representation*
Very very	$[\mu_A(x)]^4$	
More or less	$\sqrt{\mu_A(x)}$	
Somewhat	$\sqrt{\mu_A(x)}$	
Indeed	$2[\mu_A(x)]^2$ if $0 \le \mu_A \le 0.5$ $1-2[1-\mu_A(x)]^2$ if $0.5 < \mu_A \le 1$	

Fuzzy Rules

What is a fuzzy rule?

A fuzzy rule can be defined as a conditional statement in the form:

IF x is A

THEN y is B

where x and y are linguistic variables, and A and B are linguistic values determined by fuzzy sets on the universe of discourses X and Y, respectively.

What is the difference between classical and fuzzy rules?

A classical IF-THEN rule uses binary logic, for example,

Rule: 1
IF speed is >100
THEN stopping_distance is long

Rule: 2
IF speed is <40
THEN stopping_distance is short

The variable speed can have any numerical value between 0 and 220 km/h, but the linguistic variable stopping_distance can take either value long or short. In other words, classical rules are expressed in the black-and-white language of Boolean logic.

We can also represent the stopping distance rules in a fuzzy form:

Rule: 1
IF speed is fast
THEN stopping_distance is long

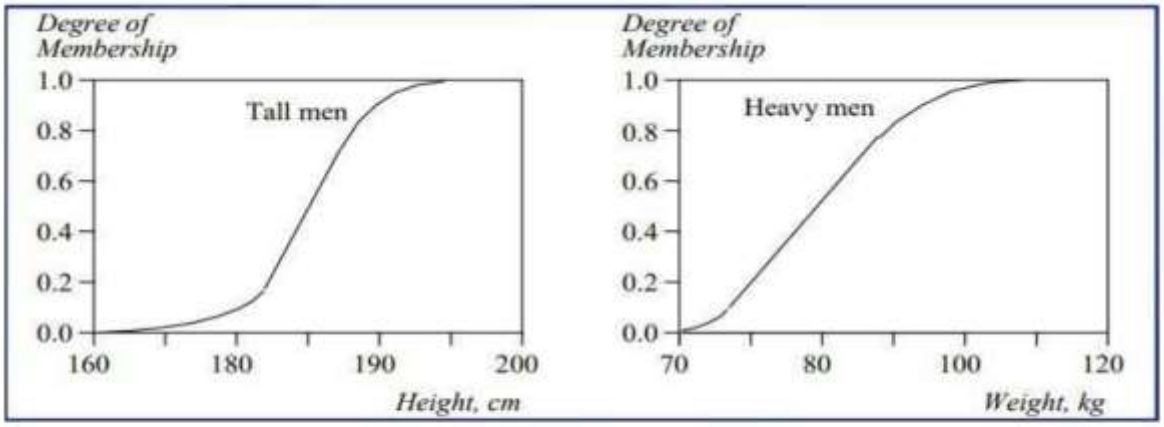
Rule: 2
IF speed is slow
THEN stopping_distance is short

In fuzzy rules, the linguistic variable speed also has the range (the universe of discourse) between 0 and 220 km/h, but the range includes fuzzy sets, such as slow, medium, and fast. The universe of discourse of the linguistic variable stopping_distance can be between 0 and 300 m and may include such fuzzy sets as short, medium and long.

Fuzzy rules relate fuzzy sets.

▪ In a fuzzy system, all rules fire to some extent, or in other words they fire partially.
If the antecedent is true to some degree of membership, then the consequent is also true to that same degree.

Fuzzy sets of tall and heavy men

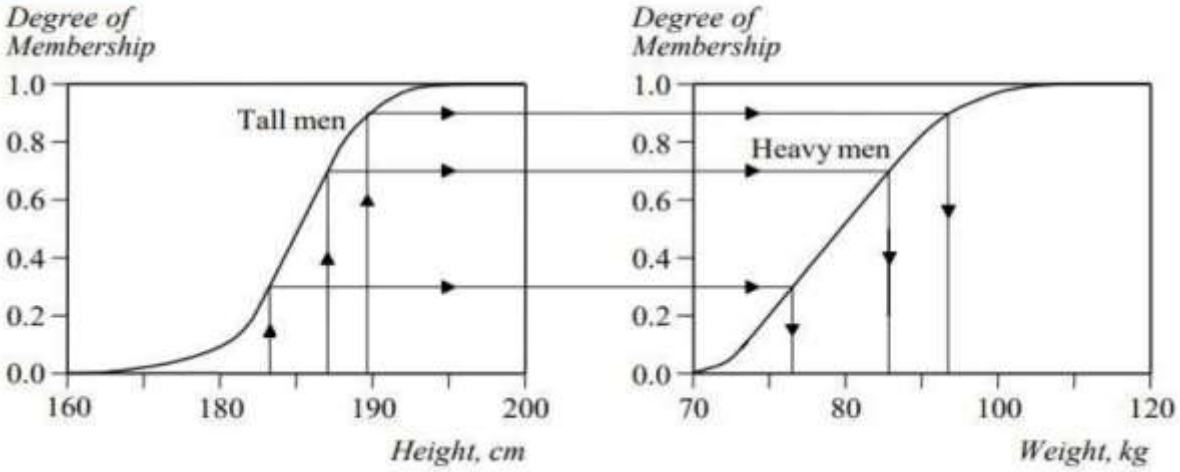


These fuzzy sets provide the basis for a weight estimation model. The model is based on a relationship between a man's height and his weight.

IF height is tall
THEN weight is heavy

12

The value of the output or a truth membership grade of the rule consequent can be estimated directly from a corresponding truth membership grade in the antecedent. This form of fuzzy inference uses a method called monotonic selection.



A fuzzy rule can have multiple antecedents, for example:

IF project_duration is long
AND project_staffing is large
AND project_funding is inadequate
THEN risk is high

IF service is excellent
OR food is delicious
THEN tip is generous

The consequent of a fuzzy rule can also include multiple parts for instance:

IF temperature is hot
THEN hot_water is reduced;
cold_water is increased

Fuzzy Proposition

Main difference between classical proposition and fuzzy proposition is in the range of their truth values. The proposition value for classical proposition is either true or false but in case of fuzzy proposition the range is not confined to only two values it varies from 2 to n. For example speed may be fast, very fast, medium, slow, and very slow. In fuzzy logic the truth value of fuzzy proposition is also depend on an additional factor known as degree of truth whose value is varies between 0 and 1. For example

p: Speed is Slow

T(p) = 0.8, if p is partly true

T(p) = 1, if p is absolutely true

T(p) = 0, if p is totally false
So, we can say that fuzzy proposition is a statement p which acquires a fuzzy truth value T(p) ranges from(0 to1).

Different types of Fuzzy Propositions

1. Unconditional and unqualified propositions

The canonical form of this type of fuzzy proposition is
p:V is F
Where, V is a variable which takes value v from a universal set U. F is a fuzzy set on U that represents a given inaccurate predicate such as fast, low, tall etc.
For example:
p: Speed (V) is high (F)
T(p) = 0.8, if p is partly true
T(p)=1, if p is absolutely true
T(p)=0, if p is totally false
Where,
T(p) = μF(v) membership grade function indicates the degree of truth of v belongs to F, its value ranges from 0 to 1.

2. Unconditional and qualified propositions

The canonical form of this type of fuzzy proposition is
p:V is F is S
Where, *V* and *F* have the same meaning and *S* is a fuzzy truth qualifier
Example:
Speed is high is very true

3. Conditional and unqualified propositions

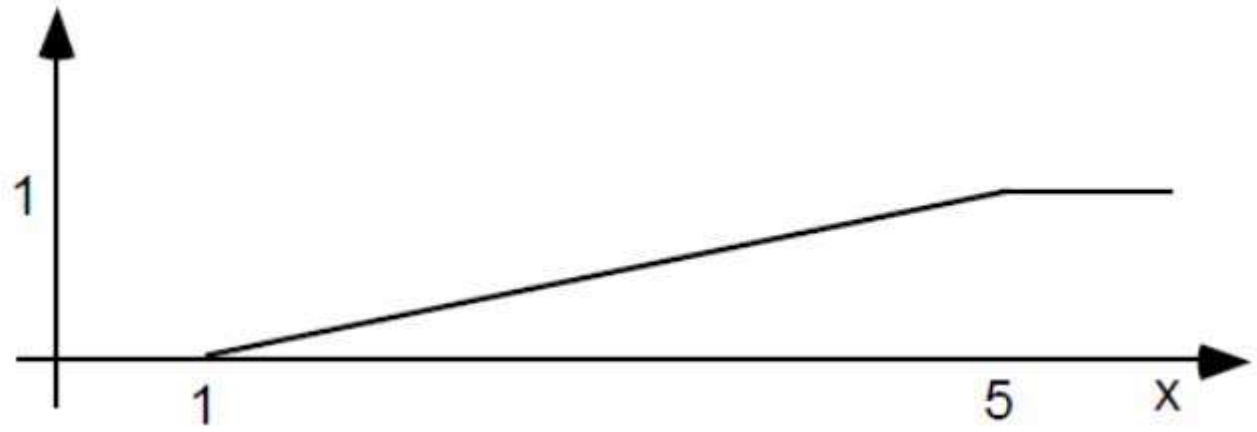
The canonical form of this type of fuzzy proposition is
p: if X is A, then Y is B
Where, *X*, *Y* are variables in universes U1 and U2
A, B are fuzzy sets on *X, Y*
Example:
p: if speed is High, then risk is Low

4. Conditional and Qualified Propositions

The canonical form of this type of fuzzy proposition is
p: (if X is A, then Y is B) is S
Where, all variables have same meaning as previous declare
Example:
p: if speed is high than risk is low is true.

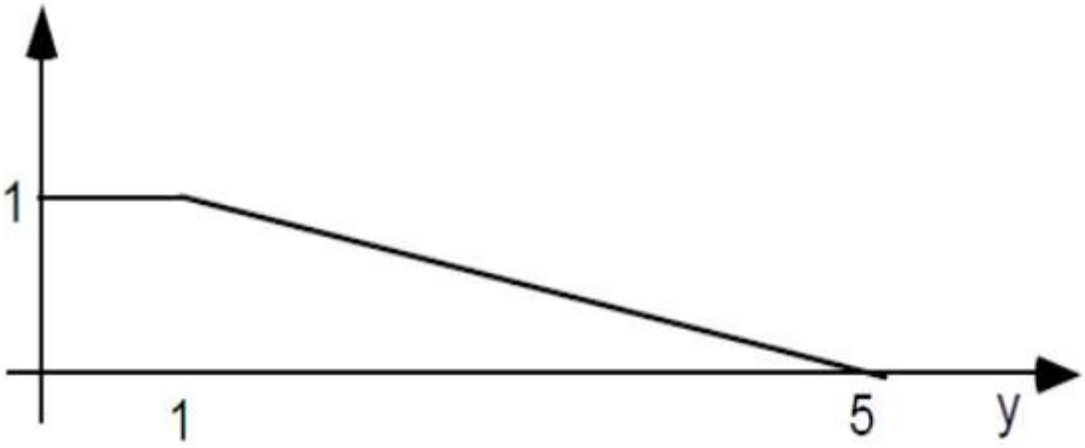
Fuzzy implication

Consider the implication statement
"if pressure is high then volume is small "
The membership function of the fuzzy set A, big pressure, illustrated in the figure



$$A(u) = \begin{cases} 1 & \text{if } u \geq 5 \\ 1 - \dfrac{5-u}{4} & \text{if } 1 \leq u \leq 5 \\ 0 & \text{otherwise} \end{cases}$$

The membership function of the fuzzy set B, small volume, can be interpreted as (see figure)



$$B(v) = \begin{cases} 1 & \text{if } v \leq 1 \\ 1 - \dfrac{v-1}{4} & \text{if } 1 \leq v \leq 5 \\ 0 & \text{otherwise} \end{cases}$$

If p: x is A
Where *A* is a fuzzy set, for example, big pressure
And q: y is B
For example, small volume
Then we define the fuzzy implication $A \rightarrow B$ as a fuzzy relation.
"IF X is A than Y is B", can be represent by relation

$$R = (A \times B) \cup (\bar{A} \times Y)$$

Where, A and B are two fuzzy sets with membership function μA and μB respectively
And Y is universe of discourse same as B but membership value for all is1.
The membership function of R is given by

$$\mu_R(x,y) = \max(\min(\mu_A(x), \mu_B(y)), 1 - \mu_A(x))$$

Example:
If X is A than Y is B
Now , suppose
X = {a,b,c}
Y = {1,2,3}
A = { (a,0) (b,0.5) (c,1) }
B = { (1,1) (2,0.3) (3,0.8)}

$$R = (A \times B) \cup (\bar{A} \times Y)$$

A X B =

	1	2	3
a	0	0	0
b	0.5	0.3	0.5
c	1	0.3	0.8

$\bar{A}$ X B =

	1	2	3
a	1	1	1
b	0.5	0.5	0.5
c	0	0	0

R =

	1	2	3
a	1	1	1
b	0.5	0.5	0.5
c	1	0.3	0.8

Fuzzy Inference
Fuzzy inference is the process of obtaining new knowledge through existing knowledge. Knowledge is most commonly represent in the form of rules or proposition for example
"if x is A then y is B" (Where A and B are linguistic values defined by fuzzy sets on universes of discourse X and Y). A rule is also called a fuzzy implication.
"x is A" is called the antecedent or premise and "y is B" is called the consequence or conclusion.
The two important inferring processes are –
- Generalized modus Ponens (GMP)
- Generalized modus Tollens (GMT)

GMP
p: If X is A than Y is B (Analytically known)
q: If X is A' (Analytically known)
than Y is B' (Analytically unknown)
Where, A, B, A', B' are fuzzy terms.
A' and B' are some predicate with different linguistic hedges. To compute the membership function of B' the min – max composition of fuzzy set A' with R(x,y) which is known as implication rule is

$$B' = A' o R(x, y)$$

In terms of membership function

where μB'(y), μA'(A'), μR(x,y) are membership function of B', A' and implication relation respectively.

$$\mu_{B'}(y) = \max(\min(\mu_{A'}(x), \mu_R(x, y)))$$

GMT

p: If X is A than Y is B (Analytically known)

q: If Y is B' (Analytically known)

---

than X is A' (Analytically unknown)

Where, A, B, A', B' are fuzzy terms.

$$A' = B' o\, R(x, y)$$

In terms of membership function

$$\mu_{A'}(y) = \max(\min(\mu_{B'}(x), \mu_R(x, y)))$$

where $\mu_{B'}(y)$, $\mu_{A'}(A')$, $\mu_R(x,y)$ are membership function of B', A' and implication relation respectively.

In terms of membership function

$$\mu_{A'}(y) = \max(\min(\mu_{B'}(x), \mu_R(x, y)))$$

where $\mu_{B'}(y)$, $\mu_{A'}(A')$, $\mu_R(x,y)$ are membership function of B', A' and implication relation respectively.

Example:

Let us consider the following logical implication,
p: If Force is huge, acceleration is large
q: If Force is huge

Acceleration is large

Let L(Large), VL(Very Large), H(Huge), VH(Very Huge) indicate the associate fuzzy variable sets.
Now let us also assume,
Force, X = {1, 2, 3, 4, 5, 6}
Acceleration, Y= {10, 20, 30, 40, 50}
L = {(3, 0.9), (4, 1), (5, 0.3)}
VL = {(5, 0.9), (6, 1)}
H = {(20, 1), (30, 0.6)}
VH = {(10, 1), (20, 0.3)}
R(x, y) = max {$H \times S$, $H \times Y$}

$$H \times L = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} 10 & 20 & 30 & 40 & 50 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0.9 & 0.6 & 0 & 0 \\ 0 & 1 & 0.6 & 0 & 0 \\ 0 & 0.3 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$\bar{H} \times Y = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} 10 & 20 & 30 & 40 & 50 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0 & 0 & 0 & 0 & 0 \\ 0.6 & 0.6 & 0.6 & 0.6 & 0.6 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$$R(x, y) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} 10 & 20 & 30 & 40 & 50 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0.2 & 0.9 & 0.6 & 0.2 & 0.2 \\ 0 & 1 & 0.5 & 0 & 0 \\ 0.6 & 0.6 & 0.6 & 0.6 & 0.6 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Now we apply the composition rule,

$$VL = VH \circ R(x, y)$$

Now we apply the composition rule,

$$VL = VH \circ R(x, y)$$

$$= [0.9 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0] \times$$

	10	20	30	40	50
1	1	1	1	1	1
2	1	1	1	1	1
3	0.2	0.8	0.5	0.2	0.2
4	0	1	0.5	0	0
5	0.6	0.6	0.6	0.6	0.6
6	1	1	1	1	1

$\therefore VL = [0.9 \quad 0.9 \quad 0.9 \quad 0.9 \quad 0.9 \quad 0.9]$

CHAPTER V: MACHINE LEARNING PARADIGMS

Topics Covered:

: Machine Learning systems, supervised and un-supervised learning, inductive learning, deductive learning, clustering, support vector machines, cased based reasoning and learning.

Machine Learning Systems:

Definition of machine learning

Arthur Samuel, an early American leader in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. He defined machine learning as "the field of study that gives computers the ability to learn without being explicitly programmed." However, there is no universally accepted definition for machine learning. Different authors define the term differently. We give below two more definitions.

1. Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience. The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data, or both (see [2] p.3).
2. The field of study known as machine learning is concerned with the question of how to construct computer programs that automatically improve with experience (see [4], Preface.).

Remarks

In the above definitions we have used the term "model" and we will be using this term at several contexts later in this book. It appears that there is no universally accepted one sentence definition of this term. Loosely, it may be understood as some mathematical expression or equation, or some mathematical structures such as graphs and trees, or a division of sets into disjoint subsets, or a set of logical "if : : : then : : : else : : :" rules, or some such thing. It may be noted that this is not an exhaustive list.

1.1.2 Definition of learning

Definition

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E.

Examples

i) Handwriting recognition learning problem
- Task T: Recognising and classifying handwritten words within images
- Performance P: Percent of words correctly classified
- Training experience E: A dataset of handwritten words with given classifications

ii) A robot driving learning problem
- Task T: Driving on highways using vision sensors
- Performance measure P: Average distance traveled before an error
- training experience: A sequence of images and steering commands recorded while observing a human driver

iii) A chess learning problem
- Task T: Playing chess
- Performance measure P: Percent of games won against opponents
- Training experience E: Playing practice games against itself

Definition

A computer program which learns from experience is called a machine learning program or simply a learning program. Such a program is sometimes also referred to as a learner.

How machines learn

1.2.1 Basic components of learning process

The learning process, whether by a human or a machine, can be divided into four components, namely, data storage, abstraction, generalization and evaluation. Figure 1.1 illustrates the various components and the steps involved in the learning process.

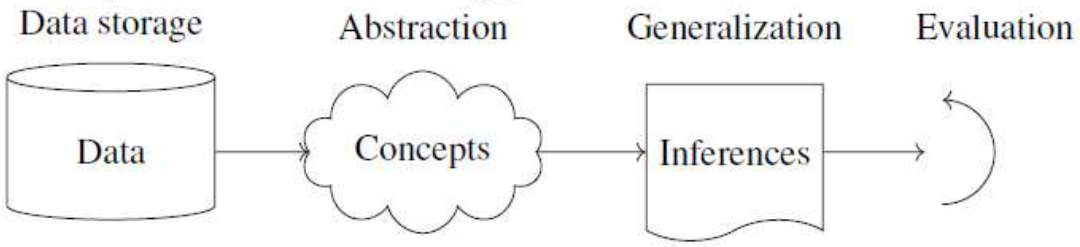


Figure 1.1: Components of learning process

1. Data storage

Facilities for storing and retrieving huge amounts of data are an important component of the learning process. Humans and computers alike utilize data storage as a foundation for advanced reasoning.

• In a human being, the data is stored in the brain and data is retrieved using electrochemical signals.

• Computers use hard disk drives, flash memory, random access memory and similar devices to store data and use cables and other technology to retrieve data.

2. Abstraction

The second component of the learning process is known as abstraction.

Abstraction is the process of extracting knowledge about stored data. This involves creating general concepts about the data as a whole. The creation of knowledge involves application of known models and creation of new models.

The process of fitting a model to a dataset is known as training. When the model has been trained, the data is transformed into an abstract form that summarizes the original information.

3. Generalization

The third component of the learning process is known as generalisation.

The term generalization describes the process of turning the knowledge about stored data into a form that can be utilized for future action. These actions are to be carried out on tasks that are similar, but not identical, to those what have been seen before. In generalization, the goal is to discover those properties of the data that will be most relevant to future tasks.

4. Evaluation

Evaluation is the last component of the learning process.

It is the process of giving feedback to the user to measure the utility of the learned knowledge. This feedback is then utilised to effect improvements in the whole learning process.

1.3 Applications of machine learning

Application of machine learning methods to large databases is called data mining. In data mining, a large volume of data is processed to construct a simple model with valuable use, for example, having high predictive accuracy.

The following is a list of some of the typical applications of machine learning.

1. In retail business, machine learning is used to study consumer behaviour.
2. In finance, banks analyze their past data to build models to use in credit applications, fraud detection, and the stock market.
3. In manufacturing, learning models are used for optimization, control, and troubleshooting.
4. In medicine, learning programs are used for medical diagnosis.
5. In telecommunications, call patterns are analyzed for network optimization and maximizing the quality of service.
6. In science, large amounts of data in physics, astronomy, and biology can only be analyzed fast enough by computers. The World Wide Web is huge; it is constantly growing and searching for relevant information cannot be done manually.
7. In artificial intelligence, it is used to teach a system to learn and adapt to changes so that the system designer need not foresee and provide solutions for all possible situations.
8. It is used to find solutions to many problems in vision, speech recognition, and robotics.
9. Machine learning methods are applied in the design of computer-controlled vehicles to steer correctly when driving on a variety of roads.
10. Machine learning methods have been used to develop programmes for playing games such as chess, backgammon and Go.

1.4 Understanding data

Since an important component of the machine learning process is data storage, we briefly consider

in this section the different types and forms of data that are encountered in the machine learning process.

1.4.1 **Unit of observation**

By a unit of observation we mean the smallest entity with measured properties of interest for a study.
Examples

- A person, an object or a thing
- A time point
- A geographic region
- A measurement

Sometimes, units of observation are combined to form units such as person-years.

1.4.2 **Examples and features**

Datasets that store the units of observation and their properties can be imagined as collections of data consisting of the following:

- Examples

An "example" is an instance of the unit of observation for which properties have been recorded. An "example" is also referred to as an "instance", or "case" or "record." (It may be noted that the word "example" has been used here in a technical sense.)

- **Features**

A "feature" is a recorded property or a characteristic of examples. It is also referred to as "attribute", or "variable" or "feature."
Examples for "examples" and "features"

1. **Cancer detection**

Consider the problem of developing an algorithm for detecting cancer. In this study we note the following.
(a) The units of observation are the patients.
(b) The examples are members of a sample of cancer patients.
(c) The following attributes of the patients may be chosen as the features:

- gender
- age
- blood pressure
- the findings of the pathology report after a biopsy

2. **Pet selection**

Suppose we want to predict the type of pet a person will choose.
(a) The units are the persons.
(b) The examples are members of a sample of persons who own pets.

features

year	model	price	mileage	color	transmission
2011	SEL	21992	7413	Yellow	AUTO
2011	SEL	20995	10926	Gray	AUTO
2011	SEL	19995	7351	Silver	AUTO
2011	SEL	17809	11613	Gray	AUTO
2012	SE	17500	8367	White	MANUAL
2010	SEL	17495	25125	Silver	AUTO
2011	SEL	17000	27393	Blue	AUTO
2010	SEL	16995	21026	Silver	AUTO
2011	SES	16995	32655	Silver	AUTO

examples

Figure 1.2: Example for "examples" and "features" collected in a matrix format (data relates to automobiles and their features)
(c) The features might include age, home region, family income, etc. of persons who own pets.

3. **Spam e-mail**

Let it be required to build a learning algorithm to identify spam e-mail.
(a) The unit of observation could be an e-mail messages.
(b) The examples would be specific messages.
(c) The features might consist of the words used in the messages.
Examples and features are generally collected in a "matrix format". Fig. 1.2 shows such a data set.

1.4.3 **Different forms of data**

1. Numeric data

If a feature represents a characteristic measured in numbers, it is called a numeric feature.

2. Categorical or nominal

A categorical feature is an attribute that can take on one of a limited, and usually fixed, number of possible values on the basis of some qualitative property. A categorical feature is also called a nominal feature.

3. Ordinal data

This denotes a nominal variable with categories falling in an ordered list. Examples include clothing sizes such as small, medium, and large, or a measurement of customer satisfaction on a scale from "not at all happy" to "very happy."

Examples

In the data given in Fig.1.2, the features "year", "price" and "mileage" are numeric and the features "model", "color" and "transmission" are categorical.

1.5 General classes of machine learning problems

1.5.1 Learning associations

1. Association rule learning

Association rule learning is a machine learning method for discovering interesting relations, called "association rules", between variables in large databases using some measures of "interestingness".

2. Example

Consider a supermarket chain. The management of the chain is interested in knowing whether there are any patterns in the purchases of products by customers like the following:
"If a customer buys onions and potatoes together, then he/she is likely to also buy hamburger."
From the standpoint of customer behaviour, this defines an association between the set of products {onion, potato} and the set {burger}. This association is represented in the form of a rule as follows:
{onion, potato} _ {burger}
The measure of how likely a customer, who has bought onion and potato, to buy burger also is given by the conditional probability
P({onion, potato}S{burger}):
If this conditional probability is 0.8, then the rule may be stated more precisely as follows:
"80% of customers who buy onion and potato also buy burger."

3. How association rules are made use of

Consider an association rule of the form
X _ Y;
that is, if people buy X then they are also likely to buy Y .
Suppose there is a customer who buys X and does not buy Y . Then that customer is a potential Y customer. Once we find such customers, we can target them for cross-selling. A knowledge of such rules can be used for promotional pricing or product placements.

4. General case

In finding an association rule X _ Y , we are interested in learning a conditional probability of the form P(Y SX) where Y is the product the customer may buy and X is the product or the set of products the customer has already purchased.
If we may want to make a distinction among customers, we may estimate P(Y SX;D) where D is a set of customer attributes, like gender, age, marital status, and so on, assuming that we have access to this information.

5. Algorithms

There are several algorithms for generating association rules. Some of the well-known algorithms are listed below:
a) Apriori algorithm
b) Eclat algorithm
c) FP-Growth Algorithm (FP stands for Frequency Pattern)

1.5.2 **Classification**

1. Definition

In machine learning, classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.

2. Example

Consider the following data:

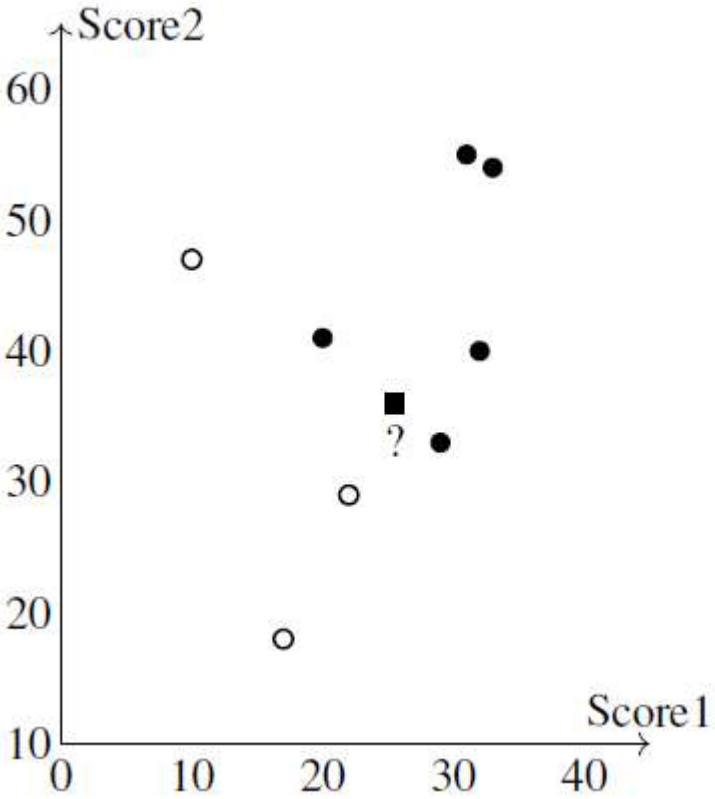
Score1	29	22	10	31	17	33	32	20
Score2	43	29	47	55	18	54	40	41
Result	Pass	Fail	Fail	Pass	Fail	Pass	Pass	Pass

Table 1.1: Example data for a classification problem

Table 1.1: Example data for a classification problem

Data in Table 1.1 is the training set of data. There are two attributes "Score1" and "Score2". The class label is called "Result". The class label has two possible values "Pass" and "Fail". The data can be divided into two categories or classes: The set of data for which the class label is "Pass" and the set of data for which the class label is"Fail".

Let us assume that we have no knowledge about the data other than what is given in the table. Now, the problem can be posed as follows: If we have some new data, say "Score1 = 25" and "Score2 = 36", what value should be assigned to "Result" corresponding to the new data; in other words, to which of the two categories or classes the new observation should be assigned? See Figure **1.4** for a graphical representation of the problem.



1.5

Figure 1.3: Graphical representation of data in Table 1.1. Solid dots represent data in "Pass" class and hollow dots data in "Fail" class. The class label of the square dot is to be determined.

To answer this question, using the given data alone we need to find the rule, or the formula, or the method that has been used in assigning the values to the class label "Result". The problem of finding this rule or formula or the method is the classification problem. In general, even the general form of the rule or function or method will not be known. So several different rules, etc. may have to be tested to obtain the correct rule or function or method.

To answer this question, using the given data alone we need to find the rule, or the formula, or the method that has been used in assigning the values to the class label "Result". The problem of

finding this rule or formula or the method is the classification problem. In general, even the general form of the rule or function or method will not be known. So several different rules, etc. may have to be tested to obtain the correct rule or function or method.

3. **Real life examples**

i) Optical character recognition

Optical character recognition problem, which is the problem of recognizing character codes from their images, is an example of classification problem. This is an example where there are multiple classes, as many as there are characters we would like to recognize. Especially interesting is the case when the characters are handwritten. People have different handwriting styles; characters may be written small or large, slanted, with a pen or pencil, and there are many possible images corresponding to the same character.

ii) **Face recognition**

In the case of face recognition, the input is an image, the classes are people to be recognized, and the learning program should learn to associate the face images to identities. This problem is more difficult than optical character recognition because there are more classes, input image is larger, and a face is three-dimensional and differences in pose and lighting cause significant changes in the image.

iii) **Speech recognition**

In speech recognition, the input is acoustic and the classes are words that can be uttered.

iv) **Medical diagnosis**

In medical diagnosis, the inputs are the relevant information we have about the patient and the classes are the illnesses. The inputs contain the patient's age, gender, past medical history, and current symptoms. Some tests may not have been applied to the patient, and thus these inputs would be missing.

v) **Knowledge extraction**

Classification rules can also be used for knowledge extraction. The rule is a simple model that explains the data, and looking at this model we have an explanation about the process underlying the data.

vi) **Compression**

Classification rules can be used for compression. By fitting a rule to the data, we get an explanation that is simpler than the data, requiring less memory to store and less computation to process.

vii) **More examples**

Here are some further examples of classification problems.

(a) An emergency room in a hospital measures 17 variables like blood pressure, age, etc. of newly admitted patients. A decision has to be made whether to put the patient in an ICU. Due to the high cost of ICU, only patients who may survive a month or more are given higher priority. Such patients are labeled as "low-risk patients" and others are labeled "high-risk patients". The problem is to device a rule to classify a patient as a "low-risk patient" or a "high-risk patient".

(b) A credit card company receives hundreds of thousands of applications for new cards. The applications contain information regarding several attributes like annual salary, age, etc. The problem is to devise a rule to classify the applicants to those who are credit-worthy, who are not credit-worthy or to those who require further analysis.

(c) Astronomers have been cataloguing distant objects in the sky using digital images created using special devices. The objects are to be labeled as star, galaxy, nebula, etc. The data is highly noisy and are very faint. The problem is to device a rule using which a distant object can be correctly labeled.

4. **Discriminant**

A discriminant of a classification problem is a rule or a function that is used to assign labels to new observations.

Examples

i) Consider the data given in Table 1.1 and the associated classification problem. We may consider the following rules for the classification of the new data:

IF Score1 + Score2 C 60, THEN "Pass" ELSE "Fail".

IF Score1 C 20 AND Score2 C 40 THEN "Pass" ELSE "Fail".

Or, we may consider the following rules with unspecified values forM;m1;m2 and then by some method estimate their values.

IF Score1 + Score2 C M, THEN "Pass" ELSE "Fail".

IF Score1 C m1 AND Score2 C m2 THEN "Pass" ELSE "Fail".

ii) Consider a finance company which lends money to customers. Before lending money, the company would like to assess the risk associated with the loan. For simplicity, let us assume that the company assesses the risk based on two variables, namely, the annual income and the annual savings of the customers.
Let x1 be the annual income and x2 be the annual savings of a customer.
• After using the past data, a rule of the following form with suitable values for _1 and _2 may be formulated:
IF x1 > _1 AND x2 > _2 THEN "low-risk" ELSE "high-risk".
This rule is an example of a discriminant.
• Based on the past data, a rule of the following form may also be formulated:
IF x2 − 0:2x1 > 0 THEN "low-risk" ELSE "high-risk".
In this case the rule may be thought of as the discriminant. The function f(x1; x2) = x2 − 0; 2x1 can also be considered as the discriminant.
5. **Algorithms**
There are several machine learning algorithms for classification. The following are some of the well-known algorithms.
a) Logistic regression
b) Naive Bayes algorithm
c) k-NN algorithm
d) Decision tree algorithm
e) Support vector machine algorithm
f) Random forest algorithm
Remarks
• A classification problem requires that examples be classified into one of two or more classes.
• A classification can have real-valued or discrete input variables.
• A problem with two classes is often called a two-class or binary classification problem.
• A problem with more than two classes is often called a multi-class classification problem.
• A problem where an example is assigned multiple classes is called a multi-label classification problem.
1**.5.3 Regression**
1. Definition
In machine learning, a regression problem is the problem of predicting the value of a numeric variable based on observed values of the variable. The value of the output variable may be a number, such as an integer or a floating point value. These are often quantities, such as amounts and sizes.
The input variables may be discrete or real-valued.
2. Example
Consider the data on car prices given in Table 1.2.

Price (US$)	Age (years)	Distance (KM)	Weight (pounds)
13500	23	46986	1165
13750	23	72937	1165
13950	24	41711	1165
14950	26	48000	1165
13750	30	38500	1170
12950	32	61000	1170
16900	27	94612	1245
18600	30	75889	1245
21500	27	19700	1185
12950	23	71138	1105

Table 1.2: Prices of used cars: example data for regression

Suppose we are required to estimate the price of a car aged 25 years with distance 53240 KM

and weight 1200 pounds. This is an example of a regression problem beause we have to predict the value of the numeric variable "Price".

3. General approach

Let x denote the set of input variables and y the output variable. In machine learning, the general approach to regression is to assume a model, that is, some mathematical relation between x and y, involving some parameters say, _, in the following form:

y = f(x; _)

The function f(x; _) is called the regression function. The machine learning algorithm optimizes the parameters in the set _ such that the approximation error is minimized; that is, the estimates of the values of the dependent variable y are as close as possible to the correct values given in the training set.

Example

For example, if the input variables are "Age", "Distance" and "Weight" and the output variable is "Price", the model may be

y = f(x; _)

Price = a0 + a1 × (Age) + a2 × (Distance) + a3 × (Weight)

where x = (Age, Distance, Weight) denotes the the set of input variables and _ = (a0; a1; a2; a3) denotes the set of parameters of the model.

4. Different regression models

There are various types of regression techniques available to make predictions. These techniques mostly differ in three aspects, namely, the number and type of independent variables, the type of dependent variables and the shape of regression line. Some of these are listed below.

• Simple linear regression: There is only one continuous independent variable x and the assumed relation between the independent variable and the dependent variable y is

y = a + bx:

• Multivariate linear regression: There are more than one independent variable, say x1; : : : ; xn, and the assumed relation between the independent variables and the dependent variable is

y = a0 + a1x1 +_+ anxn:

• Polynomial regression: There is only one continuous independent variable x and the assumed model is

y = a0 + a1x +_+ anxn:

• Logistic regression: The dependent variable is binary, that is, a variable which takes only the values 0 and 1. The assumed model involves certain probability distributions.

1.6 Different types of learning

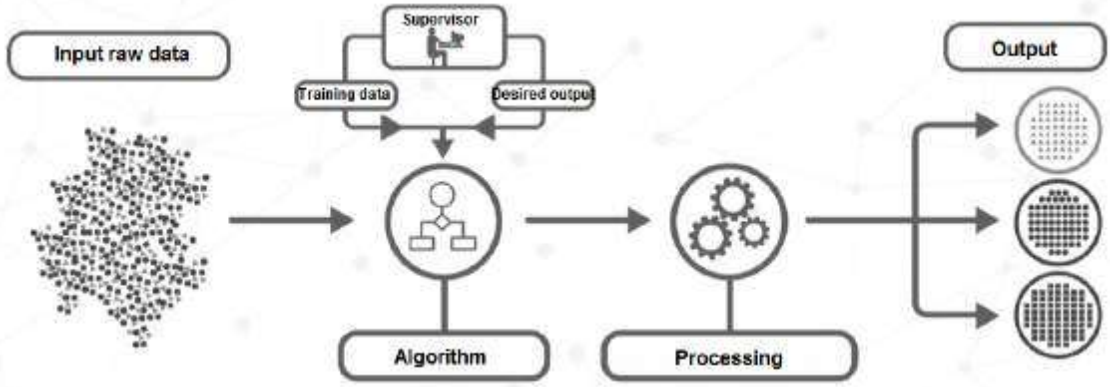
In general, machine learning algorithms can be classified into three types.

1.6.1 Supervised learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.

In supervised learning, each example in the training set is a pair consisting of an input object (typically a vector) and an output value. A supervised learning algorithm analyzes the training data and produces a function, which can be used for mapping new examples. In the optimal case, the function will correctly determine the class labels for unseen instances. Both classification and regression problems are supervised learning problems.

A wide range of supervised learning algorithms are available, each with its strengths and weaknesses. There is no single learning algorithm that works best on all supervised learning problems.



Remarks

A "supervised learning" is so called because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers (that is, the correct outputs), the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

Example

Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients and each patient is labeled as "healthy" or "sick".

gender	age	label
M	48	sick
M	67	sick
F	53	healthy
M	49	healthy
F	34	sick
M	21	healthy

Based on this data, when a new patient enters the clinic, how can one predict whether he/she is healthy or sick?

1.6.2 **Unsupervised learning**

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses.

In unsupervised learning algorithms, a classification or categorization is not included in the observations. There are no output values and so there is no estimation of functions. Since the examples given to the learner are unlabeled, the accuracy of the structure that is output by the algorithm cannot be evaluated.

The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.

Example

Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients.

gender	age
M	48
M	67
F	53
M	49
F	34
M	21

Based on this data, can we infer anything regarding the patients entering the clinic?

1.6.3 **Reinforcement learning**

Reinforcement learning is the problem of getting an agent to act in the world so as to maximize its

rewards.

A learner (the program) is not told what actions to take as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situations and, through that, all subsequent rewards.

For example, consider teaching a dog a new trick: we cannot tell it what to do, but we can reward/punish it if it does the right/wrong thing. It has to find out what it did that made it get the reward/punishment. We can use a similar method to train computers to do many tasks, such as playing backgammon or chess, scheduling jobs, and controlling robot limbs.

Reinforcement learning is different from supervised learning. Supervised learning is learning from examples provided by a knowledgeable expert.

PERSPECTIVES AND ISSUES IN MACHINE LEARNING

Perspectives in Machine Learning

One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner.

For example, consider the space of hypotheses that could in principle be output by the above checkers learner. This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights ***wo*** through ***w6.*** The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with the available training examples. The LMS algorithm for fitting weights achieves this goal by iteratively tuning the weights, adding a correction to each weight each time the hypothesized evaluation function predicts a value that differs from the training value. This algorithm works well when the hypothesis representation considered by the learner

defines a continuously parameterized space of potential hypotheses.

Issues in Machine Learning

Our checkers example raises a number of generic questions about machine learning. The field of machine learning, and much of this book, is concerned with answering questions such as the following:

☐ What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?

☐ How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?

☐ When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?

☐ What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?

☐ What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?

☐ How can the learner automatically alter **its** representation to improve its ability to represent and learn the target function?

Inductive Learning

An technique of machine learning called inductive learning trains a model to generate predictions based on examples or observations. During inductive learning, the model picks up knowledge from particular examples or instances and generalizes it such that it can predict outcomes for brand-new data.

When using inductive learning, a rule or method is not explicitly programmed into the model. Instead, the model is trained to spot trends and connections in the input data and then utilize this knowledge to

predict outcomes from fresh data. Making a model that can precisely anticipate the result of subsequent instances is the aim of inductive learning.

In supervised learning situations, where the model is trained using labeled data, inductive learning is frequently utilized. A series of samples with the proper output labels are used to train the model. The model then creates a mapping between the input data and the output data using this training data. The output for fresh instances may be predicted using the model after it has been trained.

Inductive learning is used by a number of well-known machine learning algorithms, such as decision trees, k-nearest neighbors, and neural networks. Because it enables the development of models that can accurately anticipate new data, even when the underlying patterns and relationships are complicated and poorly understood, inductive learning is an essential method for machine learning.

Advantages

- Because inductive learning models are flexible and adaptive, they are well suited for handling difficult, complex, and dynamic information.
- Finding hidden patterns and relationships in data: Inductive learning models are ideally suited for tasks like pattern recognition and classification because they can identify links and patterns in data that may not be immediately apparent to humans.
- Huge datasets – Inductive learning models are suitable for applications requiring the processing of massive quantities of data because they can efficiently handle enormous volumes of data.
- Appropriate for situations where the rules are ambiguous – Since inductive learning models may learn from examples without explicit programming, they are suitable for situations when the rules are not precisely described or understood beforehand.

Disadvantages

- May overfit to particular data – Inductive learning models that have overfit to specific training data, or that have learned the noise in the data rather than the underlying patterns, may perform badly on fresh data.
- computationally costly possible – The employment of inductive learning models in real-time applications may be constrained by their computationally costly nature, especially for complex datasets.
- Limited interpretability – Inductive learning models may be difficult to understand, making it difficult to understand how they arrive at their predictions, in applications where the decision-making process must be transparent and explicable.
- Inductive learning models are only as good as the data they are trained on, therefore if the data is inaccurate or inadequate, the model may not perform effectively.

Deductive Learning

Deductive learning is a method of machine learning in which a model is built using a series of logical principles and steps. In deductive learning, the model is specifically designed to adhere to a set of guidelines and processes in order to produce predictions based on brand-new, unexplored data.

In rule-based systems, expert systems, and knowledge-based systems, where the rules and processes are clearly set by domain experts, deductive learning is frequently utilized. The model is trained to adhere to the guidelines and processes in order to derive judgments or predictions from the input data.

Deductive learning begins with a set of rules and processes and utilizes these rules to generate predictions on incoming data, in contrast to inductive learning, which learns from particular examples. Making a model that can precisely adhere to a set of guidelines and processes in order to generate predictions is the aim of deductive learning.

Deductive learning is used by a number of well-known machine learning algorithms, such as decision trees, rule-based systems, and expert systems. Deductive learning is a crucial machine learning strategy because it enables the development of models that can generate precise predictions in accordance with predetermined rules and guidelines.

Advantages

- More effective – Since deductive learning begins with broad concepts and applies them to particular cases, it is frequently quicker than inductive learning.
- Deductive learning can sometimes yield more accurate findings than inductive learning since it starts with certain principles and applies them to the data.
- Deductive learning is more practical when data are sparse or challenging to collect since it requires fewer data than inductive learning.

Disadvantages

- Deductive learning is constrained by the rules that are currently in place, which may be insufficient or obsolete.

- Deductive learning is not appropriate for complicated issues that lack precise rules or correlations between variables, nor is it appropriate for ambiguous problems.
- Results that are biased – The quality of the rules and knowledge base, which might add biases and mistakes to the results, determines how accurate deductive learning is.

The Main Distinctions Between Inductive and Deductive Learning in Machine Learning are Outlined in the Following Table

	Inductive Learning	**Deductive Learning**
Approach	Bottom-up	Top-down
Data	Specific examples	Logical rules and procedures
Model Creation	Find correlations and patterns in data.	obey clearly stated guidelines and instructions
Training	Adapting model parameters and learning from instances	Programming explicitly and establishing rules
Goal	Using fresh data, generalizing, and making predictions.	Make a model that precisely complies with the given guidelines and instructions.
Examples	Decision trees, neural networks, clustering algorithms	Knowledge-based systems, expert systems, and rule-based systems
Strengths	capable of learning from a variety of complicated data, adaptable, and versatile	accurately when according to established norms and processes, and effective when doing specific duties
Limitation	It may be difficult to manage complex and diverse data and may overfit to specific facts.	limited to well-defined duties and norms, possibly incapable of adjusting to novel circumstances

2.2.1. Introduction to clustering

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:

"*Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.*"

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format**

Example: Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.

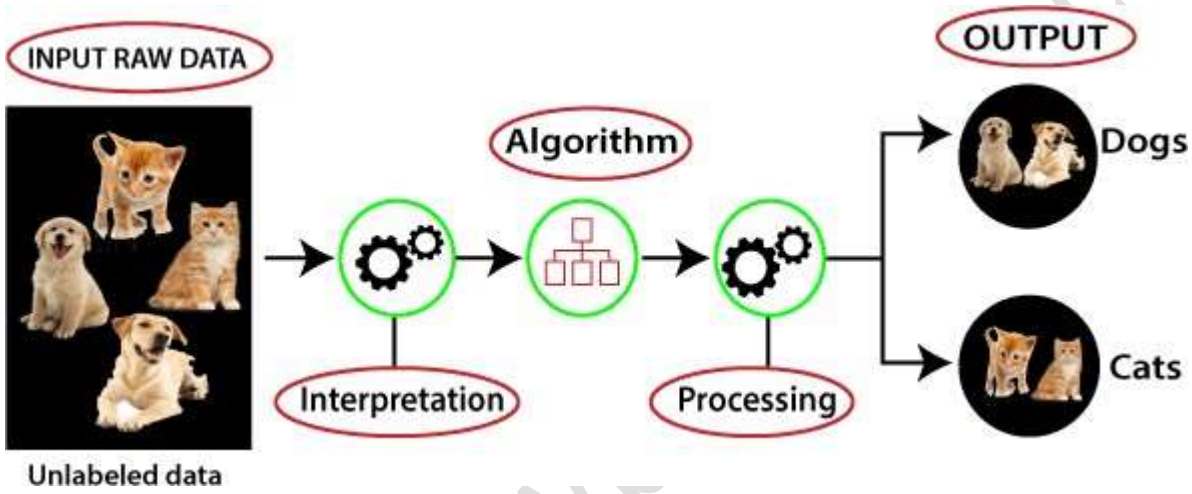
Why use Unsupervised Learning?

Below are some main reasons which describe the importance of Unsupervised Learning:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Working of Unsupervised Learning

Working of unsupervised learning can be understood by the below diagram:

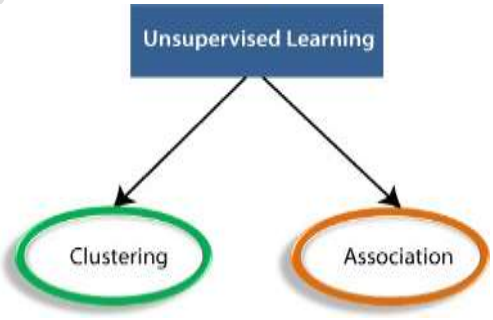


Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

Types of Unsupervised Learning Algorithm:

The unsupervised learning algorithm can be further categorized into two types of problems:



- **Clustering**: Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.

- **Association**: An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

Unsupervised Learning algorithms:

Below is the list of some popular unsupervised learning algorithms:

- K-means clustering
- KNN (k-nearest neighbors)
- Hierarchal clustering
- Anomaly detection
- Neural Networks
- Principle Component Analysis
- Independent Component Analysis
- Apriori algorithm
- Singular value decomposition

Advantages of Unsupervised Learning

- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

Disadvantages of Unsupervised Learning

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

2.2.2. K-Mean Clustering

k-means clustering algorithm

One of the most used clustering algorithm is *k-means*. It allows to group the data according to the existing similarities among them in *k* clusters, given as input to the algorithm. I'll start with a simple example.
Let's imagine we have 5 objects (say 5 people) and for each of them we know two features (height and weight). We want to group them into *k=2* clusters.

Our dataset will look like this:

	Height (H)	Weight (W)
Person 1	167	55
Person 2	120	32
Person 3	113	33
Person 4	175	76
Person 5	108	25

First of all, we have to initialize the value of the centroids for our clusters. For instance, let's choose Person 2 and Person 3 as the two centroids *c1* and *c2*, so that *c1=(120,32)* and *c2=(113,33)*.
Now we compute the euclidian distance between each of the two centroids and each point in the data. If you did all the calculations, you should have come up with the following numbers:

	Distance of object from *c1*	**Distance of object from** *c2*
Person 1	52.3	58.3
Person 2	0	7.1
Person 3	7.1	0
Person 4	70.4	75.4
Person 5	13.9	9.4

At this point, we will assign each object to the cluster it is closer to (that is taking the minimum between the two computed distances for each object).

We can then arrange the points as follows:

Person 1 → cluster 1
Person 2 → cluster 1
Person 3 → cluster 2
Person 4 →
cluster 1 Person
5→ cluster 2

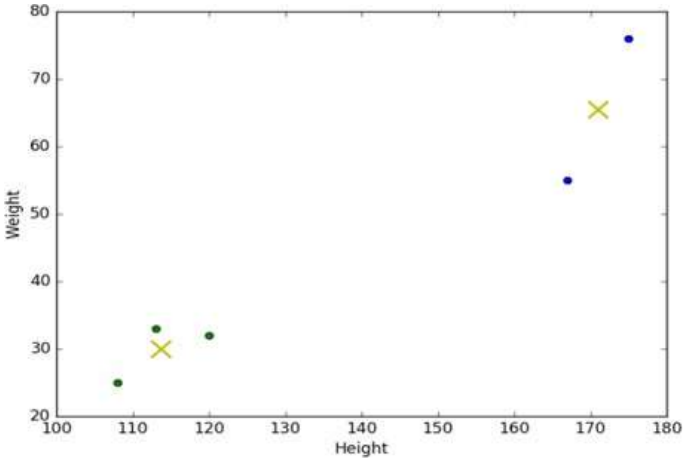
Let's iterate, which means to redefine the centroids by calculating the mean of the members of each of the two clusters.

So *c'1* = ((167+120+175)/3, (55+32+76)/3) = (154, 54.3) and *c'2* = ((113+108)/2, (33+25)/2) = (110.5, 29)

Then, we calculate the distances again and re-assign the points to the new centroids.

We repeat this process until the centroids don't move anymore (or the difference between them is under a certain small threshold).

In our case, the result we get is given in the figure below. You can see the two different clusters labelled with two different colours and the position of the centroids, given by the crosses.



How to apply k-means?

As you probably already know, I'm using Python libraries to analyze my data. The *k-means* algorithm is implemented in the *scikit-learn* package. To use it, you will just need the following line in your script:

What if our data is... non-numerical?

At this point, you will maybe have noticed something. The basic concept of *k-means* stands on mathematical calculations (means, euclidian distances). But what if our data is non-numerical or, in other words, *categorical*? Imagine, for instance, to have the ID code and date of birth of the five people of the previous example, instead of their heights and weights.
We could think of transforming our categorical values in numerical values and eventually apply *k- means*. But beware: *k-means* uses numerical distances, so it could consider close two really distant objects that merely have been assigned two close numbers.

k-modes is an extension of *k-means*. Instead of distances it uses *dissimilarities* (that is, quantification of the total mismatches between two objects: the smaller this number, the more similar the two objects). And instead of means, it uses *modes.* A mode is a vector of elements that minimizes the dissimilarities between the vector itself and each object of the data. We will have as many modes as the number of clusters we required, since they act as centroids.

Case-Based Reasoning

During the 70s and 80s, one of the most visible developments in AI research was the emergence of rule-based expert systems (RBES).
These programs were applied to more and more problem domains requiring extensive knowledge for very specific and rather critical tasks including hardware troubleshooting, geological exploration and medical diagnosis.
In general, the RBES should be based upon a deep, explicit, causal model of the problem domain knowledge that enables them to reason using first principles. But whether the knowledge is shallow or deep, an explicit model of the domain must still be elicited and implemented.
Hence, despite their success in many sectors, developers of RBES have met several critical problems.
These can be summarized as proposed by Schank [16]:
1. Difficult and time-consuming construction of the intended knowledge base due to complex and time-consuming expert knowledge elicitation. This is especially the case with problem domains covering a broad range of knowledge.
2. Incapability of dealing with problems that are not explicitly covered by the utilized rule base. In general, rule-based expert systems are useful if the built-in knowledge is well formalized, circumscribed, established and stable.
3. If no learning facility is built into a rule-based expert system, any addition to the existing program requires a programmer intervention. Solutions to these problems have been sought through better elicitation techniques and tools, improved development paradigms, knowledge modelling languages and ontologies, and advanced techniques and tools for maintaining systems. However, in the past decade an alternative reasoning paradigm and computational problem-solving method attracted a great deal of attention: Case-Based Reasoning (CBR) solves new problems by adapting previously successful solutions to similar problems. CBR draws attention because it seems to address the problems outlined above directly [19]:

• CBR does not require an explicit domain model and so elicitation becomes a task of gathering case histories.
• Implementation is reduced to identifying significant features that describe a case, an easier task than creating an explicit model.
• CBR systems can learn by acquiring new knowledge as cases. This and the application of database techniques makes the maintenance of large volumes of information easier. The work of Roger Schank [15], [16], is widely held to be the origin of CBR. He proposed a different view on model-based reasoning inspired by human reasoning and memory organization. Schank suggests that our knowledge about the world is mainly organized as memory packets holding together particular episodes from our lives that were significant enough to remember. These memory organization packets (MOPs) and their elements are not isolated but interconnected by our expectations as to the normal progress of events (called scripts by Schank). In turn, there is a hierarchy of MOPs in which "big" MOPs share "small" MOPs. If a MOP contains a situation where some problem was successfully solved and the person finds himself in a similar situation, the previous experience is recollected and the person can try to follow the same steps in order to reach a solution. Thus, rather than following a general set of rules, reapplying previously successful solution schemes in a new but similar context solves the newly encountered problems. Using these observations about human reasoning process Schank proposed memory-based reasoning model and memorybased expert systems [16] , which are characterized as follows: 10 • The utilized knowledge base is derived primarily from enumeration of specific cases or experiences. This is founded upon the observation that human experts are much more capable of recalling experiences than of articulating internal rules. • As problems are presented to a memory-based expert system to which no specific case or rule can match exactly, the system can reason from more general similarities to come up with an answer. This is founded upon the generalization power of human reasoning. In general, we are reminded of something by the similarity, but the retrieval can be also based on differences. Furthermore, the retrieval is almost never full breadth and is highly context dependent. The reason for not performing an exhaustive recall is not only due to the cumbersomeness of such a task but also due to the organizations of MOPs: once we focus on some MOP it is very easy to recall other MOPs related to it by some features. • The memory of experiences utilized by the system is changed and augmented by each additional case that is presented. A cornerstone of the memory-based model of reasoning is automatic learning: the system should remember the problems that it has encountered and use that information to solve future problems. This is founded upon the capability of the human brain to merge the progress of events seamlessly into the previously developed scripts of events. The area of AI concerned with case-based reasoning puts Schank's memory-based reasoning model in practice. In a nutshell, CBR is reasoning by remembering: previously solved problems (cases) are used to suggest solutions for novel but similar problems.

Kolodner [6] lists four assumptions about the world around us that represent the basis of the CBR approach:

1. **Regularity**: the same actions executed under the same conditions will tend to have the same or similar outcomes.
2. **Typicality**: experiences tend to repeat themselves.
3. **Consistency**: small changes in the situation require merely small changes in the interpretation and in the solution.
4. **Adaptability**: when things repeat, the differences tend to be small, and the small differences are easy to compensate for.

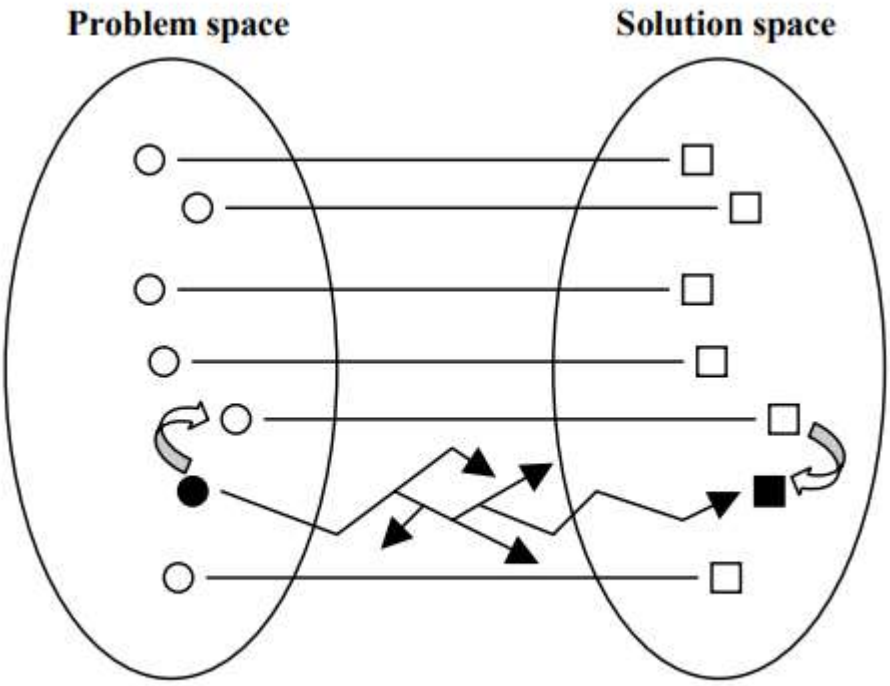


Fig.1: Problem solving using CBR

Fig. 1 illustrates how the assumptions listed above are used to solve problems in CBR. Once the currently encountered problem is described in terms of previously solved problems, the most similar solved problem can be found. The solution to this problem might be directly applicable to the current problem but, usually, some adaptation is required. The adaptation will be based upon the differences between the current problem and the problem that served to retrieve the solution. Once the solution to the new problem has been verified as correct, a link between it and the description of the problem will be created and this additional problemsolution pair (case) will be used to solve new problems in the future. Adding of new cases will improve results of a CBR system by filling the problem space more densely.
The CBR working cycle In the problem solving illustrated in Fig. 1 and explained above, the following steps were taken: describing the current problem, searching for a similar previously solved problem, retrieving the solution to it, adapting the solution to the current problem, verifying the solution, and storing the newly solved problem. In turn, since the newly found solution may be used for solving future problems, the process illustrated in Fig. 1 denotes, in fact, the CBR working cycle.

According to Kolodner, the CBR working cycle can be described best in terms of four processing stages:
1. **Case retrieval**: after the problem situation has been assessed, the best matching case is searched in the case base and an approximate solution is retrieved.
2. **Case adaptation**: the retrieved solution is adapted to fit better the new problem.
3. **Solution evaluation**: the adapted solution can be evaluated either before the solution is applied to the problem or after the solution has been applied. In any case, if the accomplished result is not satisfactory, the retrieved solution must be adapted again or more cases should be retrieved.
4. **Case-base updating**: If the solution was verified as correct, the new case may be added to the case base.

Aamodt and Plaza [1] give a slightly different scheme of the CBR working cycle comprising the four REs (Fig. 2):
1. RETRIEVE the most similar case(s);
2. REUSE the case(s) to attempt to solve the current problem;
3. REVISE the proposed solution if necessary;
4. RETAIN the new solution as a part of a new case.

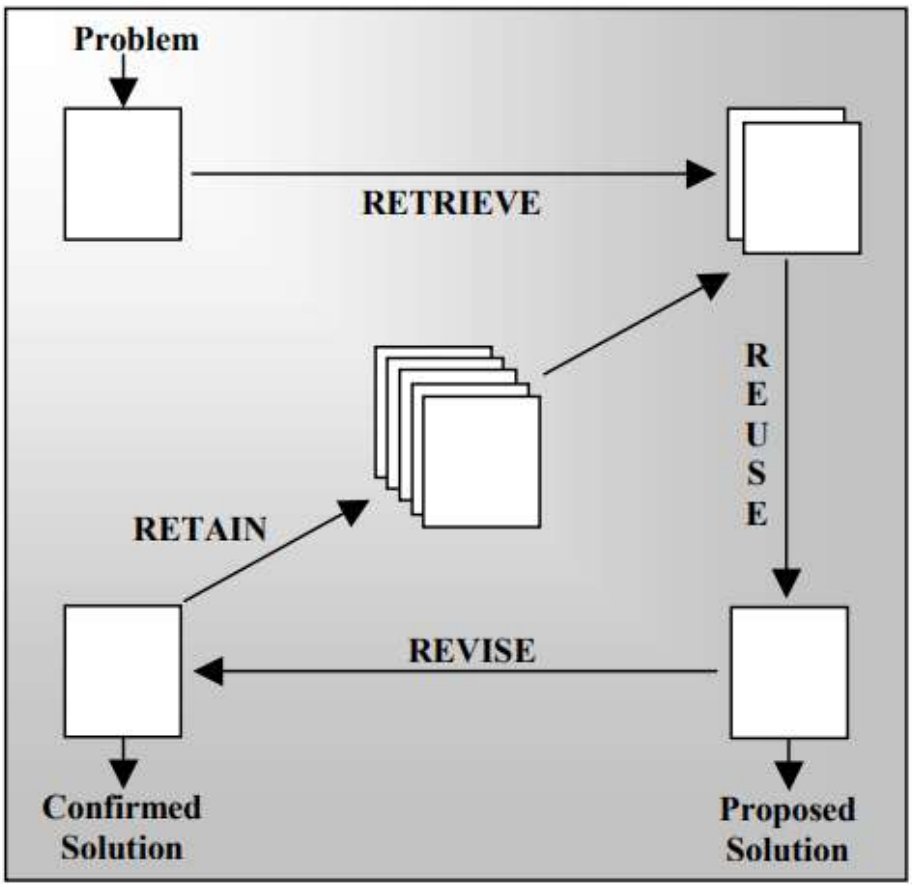


Fig. 2: The CBR cycle

Although they use different terminologies, the CBR working cycles denoted above are essentially the same. A new problem is matched against the cases furnishing the case base and one or more similar cases are retrieved. A solution suggested by the matching cases is then reused. Unless the retrieved case is a close match, the solution will probably have to be revised (adapted) and tested (evaluated) for success, producing a new case that can be retained ensuing, consequently, update of the case base.

Types of knowledge in CBR

CBR systems make use of many types of knowledge about the problem domain for which they are designed.

Richter identifies four knowledge containers [14]: **the vocabulary, similarity measures, adaptation knowledge, and cases themselves.** The first three containers usually represent general knowledge about the problem domain. If there are any exceptions from this knowledge, they are commonly handled by appropriate cases.

Vocabulary includes the knowledge necessary for choosing the features utilised to describe the cases.

Case features have to be specified so that they satisfy both: (i) being helpful in retrieving other cases, which contain useful solutions to similar problems, and (ii) being discriminative enough to prevent retrieval of too different cases, which could lead to false solutions and/or reduced performance.

A thorough comprehension of the problem domain is necessary to be able to choose which of all problem parameters are the best as case features. In addition, either the vocabulary should be chosen such that it anticipates future expansion of the case base, or the system should be developed such that it alleviates automatic expansion of the vocabulary. Otherwise, it may be impossible to represent new problem features, which will then either be mapped to the available descriptors or be ignored, probably leading in both cases to wrong solutions. Similarity measures include the knowledge about the similarity measure itself and the knowledge used to choose the most efficient organisation of the employed case base and the most appropriate case-retrieval method. For any given problem, there are many possible similarity measures that can be used. Hence, choosing the best among the available possibilities and implementing the chosen similarity measure efficiently exacts sound knowledge of the problem domain. This is especially important for classification problems involving complex structured cases since the value of the similarity can be used as a basis for automatic classification. As far as the organisation of the employed case base and the retrieval algorithm are concerned, a balance has to be found between case-memory models that preserve the semantic richness of cases and methods that simplify the access and retrieval of relevant cases. In general, knowledge about cases can be used to choose the organisational structure of the case base such that the cases can be accurately and efficiently retrieved. Adaptation knowledge includes the knowledge necessary for implementing the adaptation and evaluation stages of the CBR working

cycle. Generally, the adaptation stage requires knowledge about how differences in problems affect the solutions. This knowledge is usually coded in explicit rules. Yet, since for many problem domains, this is the most difficult knowledge to acquire, the adaptation is frequently left to the user of the system. This is especially the case when mistakes made by the system are expensive effecting the reliability of the system and, in turn, the user's confidence in it. Usually, before applying a new solution for solving a problem, its correctness has to be evaluated. The knowledge required for the evaluation stage concerns estimating the significance of differences and similarities between 13 the situations. Thus, this type of knowledge can be viewed as an extension and refinement of the knowledge furnishing the similarity measures container. Cases contain knowledge about solved problem instances and, in many CBR systems, this represents the knowledge that the system acquires during use. What the cases will contain is mainly determined by the chosen vocabulary. Sometimes the employed case base is initialized with carefully selected cases that provide a problem domain coverage that is as even as possible (e.g., this is the case with the Facial Expression Analyzer described in [11]). This is commonly the case when the necessary adaptation stage is to be kept simple, yielding manageable system maintenance. Anyhow, new cases will usually be added during use. Yet, it is often unwise to store all the solved problems as cases. Large case bases may have high memory/storage requirements, may impose long retrieval times and, in turn, may reduce the system's performance. Therefore, heuristics should be specified for determining the useful cases to be stored in the case base.

Case representation:

A case is a contextualized piece of knowledge representing an experience. It contains the past lesson that is the content of the case and the context in which the lesson can be used. In general, a case comprises a:

- Problem description, which depicts the state of the world when the case occurred;
- Problem solution which states the derived solution to that problem; and/or
- Outcome, which describes the state of the world after the case occurred. Cases that comprise problems and their solutions can be used to derive solutions to new problems, whereas cases comprising problems and outcomes can be used to evaluate new situations. If such cases contain solutions in addition, they can be used to evaluate the outcome of proposed solutions and prevent potential problems. The more information is stored, the more useful the case can be. Yet entering all available information makes the system more complex and, in turn, more difficult to use. Due to these reasons, most of the CBR systems are limited to storing only problem descriptions and solutions. The problem description essentially contains as much data about the problem and its context as necessary for an efficient and accurate case retrieval. Principally, it is useful to store retrieval statistics like the number of times the case was retrieved and the average match value. These statistics may be valuable for handling the case base: for prioritising cases, for pruning the case base by removing seldom used cases, and generally for maintenance of the case base. The problem solution can be either atomic or compound. Atomic solutions are typical for CBR systems used for diagnosis or for classification in general. Compound solutions can be found for instance in CBR systems utilised for planning or design. A compound solution may be composed of a sequence of actions, an arrangement of components, etc. For example, in the case of the Facial Expression Analyzer reported in [11], compound solutions consist of multiple, user-defined, facial-expression interpretation labels. The main use of a solution is to serve as a starting point for educing new solutions. Therefore, the way a solution is derived may be of equal importance as that of the solution itself.

Cases can be represented as simple feature vectors, or they can be represented using any AI representational formalism such as frames, objects, predicates, semantic nets, or rules. The choice of particular representational formalism is largely determined by the information to be stored within a case. Cases can be monolithic or compound. Individual parts of compound cases can be processed or used separately. For example, a problem can be solved by reusing partial solutions from several compound cases, like explained in [11]. Most representational formalisms are proprietary for the more complex cases. Nevertheless, there is a lack of consensus within the CBR community as to exactly what information should be stored within a case and, in turn, which representational formalism should be used. However, two pragmatic measures can be taken into account in deciding both the information to be stored in a case and the appropriate representational formalism: the intended functionality and the ease of acquisition of the information represented in the case. Finally, cases are the basis of any CBR system: a system without cases would not be a casebased system. Yet, a system using only cases and no other explicit knowledge (not even in the similarity measures) is difficult to distinguish from a nearest-neighbour classifier or a database retrieval system. In other words, such a system does not exploit the full generalisation power of CBR, resulting usually in poor system performance due to inefficient retrieval based upon case-by-case search of the whole case base.

Indexing Within the CBR community, an explicit formal specification (i.e. ontology) of what the terms "indices" and "indexing" actually mean in terms of a CBR system has not been established yet. Kolodner identifies indexing with an accessibility problem [6], that is, with the whole set of issues inherent in setting

up the case base and its retrieval process so that the right cases are retrieved at the right time. Thus, case indexing involves assigning indices to cases to facilitate their retrieval. CBR researches proposed several guidelines on indexing [19].
Indexes should be:
• predictive of the case relevance;
• recognisable in the sense that it should be understandable why they are used;
• abstract enough to allow for widening the future use of the case base;
• concrete (discriminative) enough to facilitate efficient and accurate retrieval. Both manual and automated methods are used nowadays to select indices. Choosing indices manually involves deciding the purpose of a case with respect to the aims of the user and deciding under which circumstances the case will be useful. Kolodner claims that people tend to be better at choosing the indices than automatic algorithms. Anyhow, there is an ever increasing number of automated indexing methods. For a review of these the reader is referred to [19]. For an example of an automatic indexing algorithm performing indexing cases by (case) features that tend to be predictive across the entire problem domain, the reader is referred to [11]. Indices do not have to be rigid; they may change during use of the system. In fact, changing the indexes is one way of learning. Changes may be made if, for instance, a wrong case was retrieved or an entirely novel problem situation is encountered. Changes may involve changing weights (importance/priority) of the features, changing or adding features, changing 15 or adding pointers to other cases in the case base, etc. Similarly to selecting/generating the indexes, changing the indexes can be done either manually or automatically.

Case base organization Case storage is an important aspect in designing efficient CBR system, in that it should reflect the conceptual view of what is represented in the case and take into account the indexes that characterise the case. As already mentioned above, the case base should be organised into a manageable structure that supports efficient and accurate search and retrieval methods. Accurate retrieval guarantees that the best matching case will be retrieved, and efficient retrieval guarantees that cases will be retrieved fast enough for acceptable system response times. These two factors are inversely proportional: it is easy to guarantee accurate retrieval at the expense of efficiency (e.g. by matching all the cases) and easy to have fast retrieval if only a fraction of the employed case base is searched (possibly missing some examples). Hence, a good case-base organisation and a good retrieval algorithm are the ones which yield the best compromise between accuracy and efficiency of the retrieval algorithm. In general, three main approaches to case-base organisation can be distinguished: flat organisation, clustered organisation, and hierarchical organisation. Also a combination of these methods within the same case base is possible. Flat organisation is the simplest case-base organisation that yields a straightforward flat structure of the case base. Though advantageous due to its simplicity and facile case addition/deletion, a flat case-base organisation imposes, in general, case retrieval based upon a case-by-case search of the whole case base. Hence, for medium and large case bases, this leads to time-consuming retrieval, yielding an inefficient CBR system.
Clustered organisation, originating in the dynamic memory model initially proposed by Schank [15], is the type of case-base organisation in which cases are stored in clusters of similar cases. The grouping of cases may be based on their mutual similarity (like in the case of the dynamic memory of experiences used by Pantic [10], [11]) or on the similarity to some prototypical cases. The advantage of this organisation is that the selection of the clusters to be matched is rather easy, as it is based upon the indexes and/or prototypical cases characterising the clusters.
A disadvantage is that it needs a more complex algorithm for case addition/ deletion than a flat organised case base. Hierarchical organisation, originating in the category-exemplar memory model of Porter and Bareiss [12], is the case-base organisation that is generally obtained when cases that share the same features are grouped together.
The case memory is a network structure of categories, semantic relations, cases, and index pointers. Each case is associated with a category, while the categories are inter-linked within a semantic network containing the features and intermediate states referred to by other terms. Different case features are assigned different importance in describing the membership of a case to a category. It is important to note that this importance assignment is static; if it changes, the case-base hierarchy has to be redefined. A new case is stored by searching for a matching case and by establishing the relevant feature indexes. If a case is found with only minor differences to the new case, the new case is usually not retained. In turn, a hierarchical case-base organisation facilitates fast and accurate case retrieval. However, its higher complexity implies a rather cumbersome case 16 addition/deletion, potentially involving expensive case-base reorganisation and an inapt casebase evaluation and maintenance. 2.6 Retrieval Given a description of a problem, a retrieval algorithm should retrieve cases that are most similar to the problem or situation currently presented to the pertinent CBR system. The retrieval algorithm relies on the indices and the organisation of the case memory to direct the search to case(s) potentially useful for solving the currently encountered problem. The issue of choosing the best matching cases can be referred to as analogy drawing, that is, comparing cases in order to determine the degree of similarity between them. Many

retrieval algorithms have been proposed in the literature up to date: induction search (e.g., ID3, [13]), nearest neighbour search, serial search, hierarchical search, parallel search, etc. (for examples, see [8]). The simplest form of retrieval is the 1st-nearest-neighbour search of the case base, which performs similarity matching on all the cases in the case base and returns just one best match [8]. It is to be expected that this method implies long retrieval times, especially in the case of large case bases. Therefore, cases are usually preselected prior to similarity matching. Cases can be preselected using a simpler similarity measure; commonly, this is done using the indexing structure of the case base. A typical problem with preselection concerns handling a situation where no best match has been found in the preselected set of cases; since preselection is merely approximate, there is a possibility that amongst the non-selected cases a better match can be found. Another way of speeding up the retrieval is to employ ranking of cases. The simplest ranking method concerns exploiting the retrieval statistics for cases in the case base. The frequently retrieved cases can be considered as prototypical cases and probably should be matched first. Another ranking method is applicable to the clustered case-base organisation. It concerns matching the current case to the clusters' prototypes and then searching the matching clusters in the order determined by the degree of similarity between the matching clusters' prototypes and the current case. The retrieval may result in retrieving single or multiple best match cases. In general, the retrieval mechanism tends to be simpler and faster if: (i) a larger number of possibly similar cases are retrieved, (ii) all of them are used to find solutions, and then (iii) the best solution is chosen. In this case, the retrieval algorithm itself may be less selective (and, therefore, simpler and faster) since the usefulness of the retrieved cases is to be determined in succeeding processing phases. Finally, a way of speeding up the retrieval is to do it in parallel. A parallel search of the case base is realisable since case matching does not require exchange of much information between the parallel running processes. Thus, the speed gain scales up with the number of processing units. While the implementation of parallel retrieval is simple for flat and clustered case bases, it is rather difficult for hierarchical case bases. Though bringing significant speed gains, parallel retrieval is usually accompanied by an increase in implementation costs and software complexity.

Adaptation Generally, once a matching case is retrieved, it will not correspond to exactly the same problem as the problem for which the solution is currently being sought. Consequently, the solution belonging to the retrieved case may not be optimal for the problem presently encountered and, therefore, it should be adapted. Adaptation looks for prominent differences between the retrieved case and the current case, and then (most commonly) applies a formulae or a set of rules to account for those differences when suggesting a solution. In general, there are two kinds of adaptation in CBR [19]: 1. Structural adaptation applies adaptation rules directly to the solution stored in cases. If the solution comprises a single value or a collection of independent values, structural adaptation can include modifying certain parameters in the appropriate direction, interpolating between several retrieved cases, voting, etc. However, if there are interdependencies between the components of the solution, structural adaptation requires a thorough comprehension and a well-defined model of the problem domain. 2. Derivational adaptation reuses algorithms, methods, or rules that generated the original solution to produce a new solution to the problem currently presented to the system. Hence, derivational adaptation requires the planning sequence that begot a solution to be stored in memory along with that solution. This kind of adaptation, sometimes referred to as reinstantiation, can only be used for problem domains that are well understood. An ideal set of rules must be able to generate complete solutions from scratch, and an effective and efficient CBR system may need both structural adaptation rules to adapt poorly understood solutions and derivational mechanisms to adapt solutions of cases that are well understood. However, one should be aware that complex adaptation procedures make the system more complex but not necessarily more powerful. Complex adaptation procedures make it more difficult to build and maintain CBR systems and may also reduce system reliability and, in turn, user's confidence in the system if faulty adaptations are encountered due to, for example, incompleteness of the adaptation knowledge, which is the most difficult kind of knowledge to acquire [7]. Therefore, in many CBR systems, adaptation is done by the user rather than by the system. Mark et al. report that in a well-designed system, the users do not perceive "manual" adaptation as something negative [7].

advantages and limitations of CBR

CBR is a lazy problem-solving method and shares many characteristics with other lazy problem-solving methods, including advantages and disadvantages.

Aha [2] defines the peculiarities of lazy problem-solving methods in terms of three Ds:

- Defer: lazy problem solvers simply store the presented data and generalizing beyond these data is postponed until an explicit request is made.
- Data-driven: lazy problem solvers respond to a given request by combining information from the stored data.
- Discard: lazy problem solvers dismiss any temporary (intermediate) result obtained during the problem solving process. Unlike lazy problem solvers, eager problem-solving methods try to extract as much

information as possible from the presented data and then to discard the data prior to the actual 18 problem solving.

An example of a lazy problem solver is a CBR classifier, while an ANN classifier is an example of an eager problem solver. Eager algorithms can be referred to as knowledge compilers, as opposed to lazy algorithms, which perform run-time knowledge interpretation.

This is the key difference between lazy and eager problem solvers, which can also be explained by the following: • Lazy methods can consider the current query instance x when deciding how to generalise beyond the training data (which have already been presented).

• Eager methods cannot, because their global approximation to the target function has already been chosen by the time they observe the current query instance x. To summarise, lazy methods have the option of selecting a different hypothesis or local approximation to the target function for each presented query instance. Eager methods using the same hypothesis space are more restricted because they must choose their approximation before the presented queries are observed. Consequently, a lazy method will generally require less computation during training, but more computation when it must generalise from training data by choosing a hypothesis based on the training examples near the currently presented query.

The benefits of CBR as a lazy problem-solving method are: • Ease of knowledge elicitation: Lazy methods, in general, can utilise easily available cases or problem instances instead of rules that are difficult to extract. So, classical knowledge engineering is replaced by case acquisition and structuring [2].

• Absence of problem-solving bias: Because cases are stored in a "raw" form, they can be used for multiple problem-solving purposes. This in contrast to eager methods, which can be used merely for the purpose for which the knowledge has already been compiled.

• Incremental learning: A CBR system can be put into operation with a minimal set of solved cases furnishing the case base. The case base will be filled with new cases as the system is used, increasing the system's problem-solving ability. Besides simple augmentation of the case base, new indexes and clusters/categories can be created and the existing ones can be changed. This in contrast to virtually all machine-learning methods (see part 1 of this syllabus), which require a special training period whenever information extraction (knowledge generalisation) is performed. Hence, dynamic on-line adaptation to a non-rigid environment is possible [8].

• Suitability for complex and not-fully formalised solution spaces: CBR systems can be applied to an incomplete model of problem domain; implementation involves both to identify relevant case features and to furnish, possibly a partial case base, with proper cases. In general, because they can handle them more easily, lazy approaches are often more appropriate for complex solution spaces than eager approaches, which replace the presented data with abstractions obtained by generalisation.

• Suitability for sequential problem solving: Sequential tasks, like these encountered in reinforcement learning problems, benefit from the storage of history in the form of a sequence of states or procedures. Such a storage is facilitated by lazy approaches.

• Ease of explanation: The results of a CBR system can be justified based upon the similarity of the current problem to the retrieved case(s). Because solutions generated by CBR are easily traceable to precedent cases, it is also easier to analyse failures of the system. As noted by Watson and Marir [19], the explanations provided based upon individual and generalised cases tend to be more satisfactory than explanations generated by chains of rules.

• Ease of maintenance: This is particularly due to the fact that CBR systems can adapt to many changes in the problem domain and the pertinent environment, merely by acquiring 19 new cases. This eliminates some need for maintenance; only the case base(s) needs to be maintained. Major disadvantages of lazy problem solvers are their memory requirements and time consuming execution due the processing necessary to answer the queries.

The limitations of CBR can be summarised as follows: • Handling large case bases: High memory/storage requirements and time-consuming retrieval accompany CBR systems utilising large case bases. Although the order of both is at most linear with the number of cases, these problems usually lead to increased construction costs and reduced system performance. Yet, these problems are less and less significant as the hardware components become faster and cheaper. • Dynamic problem domains: CBR systems may have difficulties in handling dynamic problem domains, where they may be unable to follow a shift in the way problems are solved, since they are usually strongly biased towards what has already worked. This may result in an outdated case base. • Handling noisy data: Parts of the problem situation may be irrelevant to the problem itself. Unsuccessful assessment of such noise present in a problem situation currently imposed on a CBR system may result in the same problem being unnecessarily stored numerous times in the case base because of the difference due to the noise. In turn this implies inefficient storage and retrieval of cases. • Fully automatic operation: In a typical CBR system, the problem domain is usually not fully covered. Hence, some problem situations can occur for which the system has no solution. In such situations, CBR systems commonly expect input from the user.

CBR application domains

Although CBR is a relatively new AI methodology, numerous successful applications exist in the academic as well as in the commercial domain. Already in 1994, Watson and Marir reported over 100 commercially available CBR applications [19]. The domains of these numerous CBR systems reported in the literature are the following: • Interpretation as a process of evaluating situations/problems in some context (e.g., HYPO for interpretation of patent laws proposed in 1991, KICS for interpretation of building regulations proposed in 1994, LISSA for interpretation of non-destructive test measurements proposed in 1999). • Classification as a process of explaining a number of encountered symptoms (e.g., CASEY for classification of auditory impairments proposed in 1989, CASCADE for classification of software failures proposed in 1992, PAKAR for causal classification of building defects proposed in 1994, ISFER for classification of facial expressions into userdefined interpretation categories proposed in [10]). • Design as a process of satisfying a number of posed constraints (e.g., JULIA for meal planning proposed in 1992, Déjà Vu for control-software production proposed in 1996, CLAVIER for design of optimal layouts of composite airplane parts proposed in 1996, EADOCS for aircraft panels design proposed 1997). • Planning as a process of arranging a sequence of actions in time (e.g., BOLERO for building diagnostic plans for medical patients proposed in 1993, TOTLEC for manufacturing planning proposed in 1993). • Advising as a process of resolving diagnosed problems (e.g., DECIDER for advising students proposed in 1987, HOMER – a CAD/CAM help desk proposed in 1998).

CHAPTER VI: ARTIFICIAL NEURAL NETWORKS

Topics covered:

Artificial Neural Networks, Single-Layer feed forward networks, multi-layer feed-forward networks, radial basis function networks, design issues of artificial neural networks and recurrent networks

Artificial Neural Network

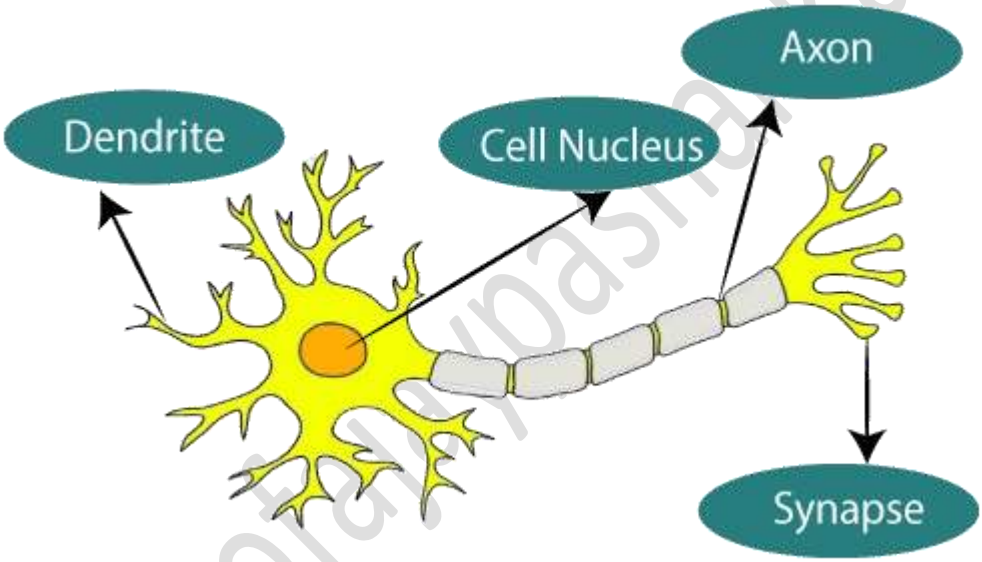
Artificial Neural Network Tutorial provides basic and advanced concepts of ANNs. Our Artificial Neural Network tutorial is developed for beginners as well as professions.

The term "Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modeled after the brain. An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.

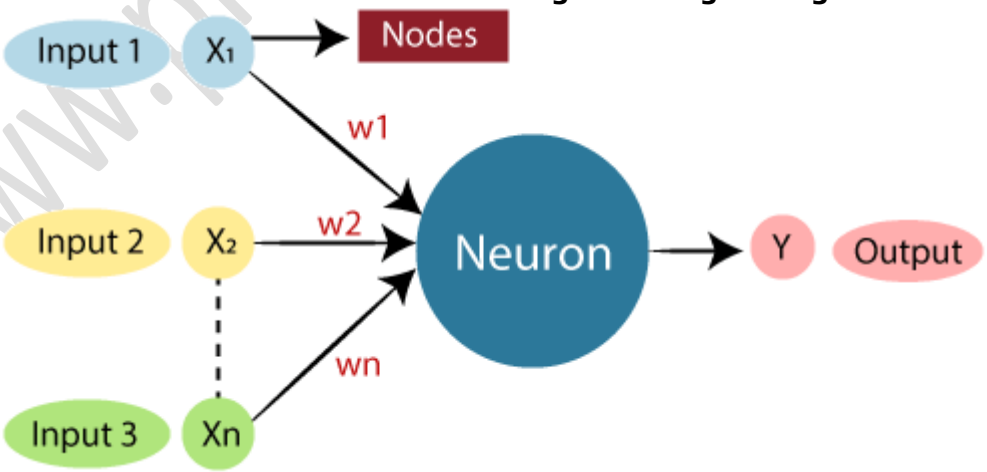
Artificial neural network tutorial covers all the aspects related to the artificial neural network. In this tutorial, we will discuss ANNs, Adaptive resonance theory, Kohonen self-organizing map, Building blocks, unsupervised learning, Genetic algorithm, etc.

What is Artificial Neural Network?

The term "**Artificial Neural Network**" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.



The given figure illustrates the typical diagram of Biological Neural Network.
The typical Artificial Neural Network looks something like the given figure.



Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

Relationship between Biological neural network and artificial neural network:

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

Human Brain v/s Artificial Neural Network Comparison between biological and artificial neurons based on the following criteria

1. **Speed** – Signals in human brain move at a speed dependent on the nerve impulse. The biological neuron is slow in processing as compared to the artificial neural networks which are modelled to process faster.
2. **Processing**- The biological neuron can perform massive parallel operations simultaneously. A large number of simple units are organized to solve problems independently but collectively. The artificial neurons also respond in parallel but do not execute programmed instructions.
3. **Size and Complexity**- The size and complexity of the brain is comparatively higher than that of artificial neural network. The size and complexity of an ANN is different for different applications
4. **Storage Capacity** – The biological neuron stores the information in its interconnection and in artificial neuron it is stored in memory locations.
5. **Tolerance**- The biological neuron has fault tolerant capability but artificial neuron has no tolerant capability. Biological neurons considers redundancies whereas artificial neurons cannot consider redundancies.
6. **Control mechanism**- There is no control unit to monitor the information processed in to the network in biological neural networks whereas in artificial neuron model all activities are continuously monitored by a control unit.

Characteristics of Artificial Neural Networks

1. It is a mathematical model consists of computational elements implemented neutrally.
2. Large number of highly interconnected processing elements known as neurons are prominent in ANN
3. The interconnections with their weights are associated with neurons.
4. The input signals arrive at the processing elements through connections and weights.
5. ANNs collective behavior is characterized by their ability to learn, recall and generalize from the given data.
6. A single neuron carries no specific information.

An **Artificial Neural Network** in the field of **Artificial intelligence** where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

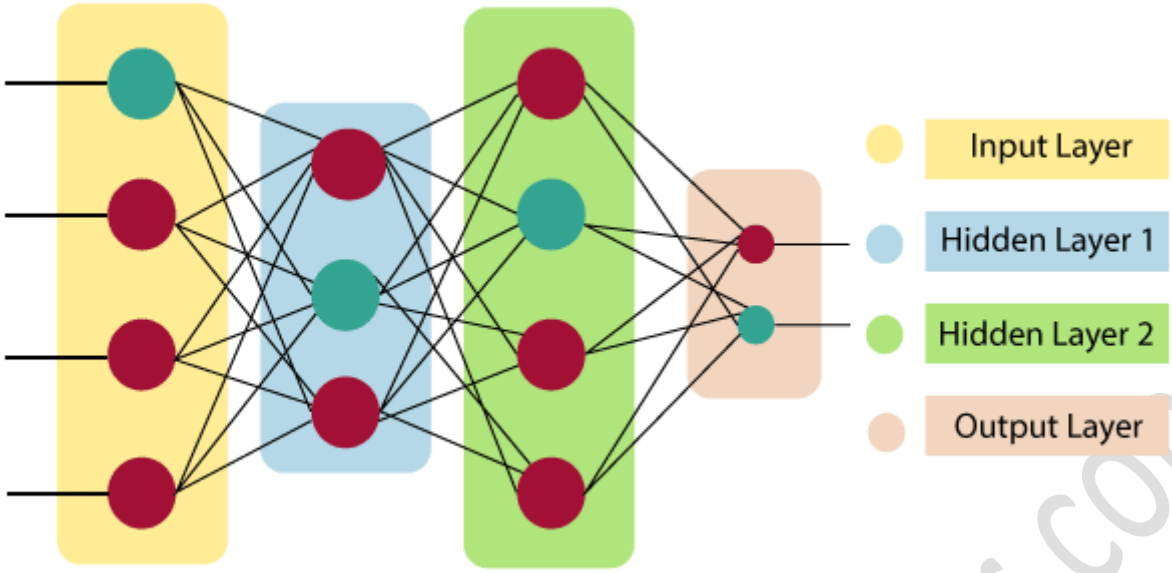
There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.

We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input. Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."

The architecture of an artificial neural network:

To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of. In order to define a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers. Lets us look at various types of layers available in an artificial neural network.

Artificial Neural Network primarily consists of three layers:



Input Layer:
As the name suggests, it accepts inputs in several different formats provided by the programmer.
Hidden Layer:
The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.
Output Layer:
The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.
The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^{n} Wi * Xi + b$$

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.
Advantages of Artificial Neural Network (ANN)
Parallel processing capability:
Artificial neural networks have a numerical value that can perform more than one task simultaneously.
Storing data on the entire network:
Data that is used in traditional programming is stored on the whole network, not on a database. The disappearance of a couple of pieces of data in one place doesn't prevent the network from working.
Capability to work with incomplete knowledge:
After ANN training, the information may produce output even with inadequate data. The loss of performance here relies upon the significance of missing data.
Having a memory distribution:
For ANN is to be able to adapt, it is important to determine the examples and to encourage the network according to the desired output by demonstrating these examples to the network. The succession of the network is directly proportional to the chosen instances, and if the event can't appear to the network in all its aspects, it can produce false output.
Having fault tolerance:
Extortion of one or more cells of ANN does not prohibit it from generating output, and this feature makes the network fault-tolerance.
Disadvantages of Artificial Neural Network:
Assurance of proper network structure:
There is no particular guideline for determining the structure of artificial neural networks. The appropriate network structure is accomplished through experience, trial, and error.
Unrecognized behavior of the network:
It is the most significant issue of ANN. When ANN produces a testing solution, it does not provide insight concerning why and how. It decreases trust in the network.
Hardware dependence:

Artificial neural networks need processors with parallel processing power, as per their structure. Therefore, the realization of the equipment is dependent.

Difficulty of showing the issue to the network:

ANNs can work with numerical data. Problems must be converted into numerical values before being introduced to ANN. The presentation mechanism to be resolved here will directly impact the performance of the network. It relies on the user's abilities.

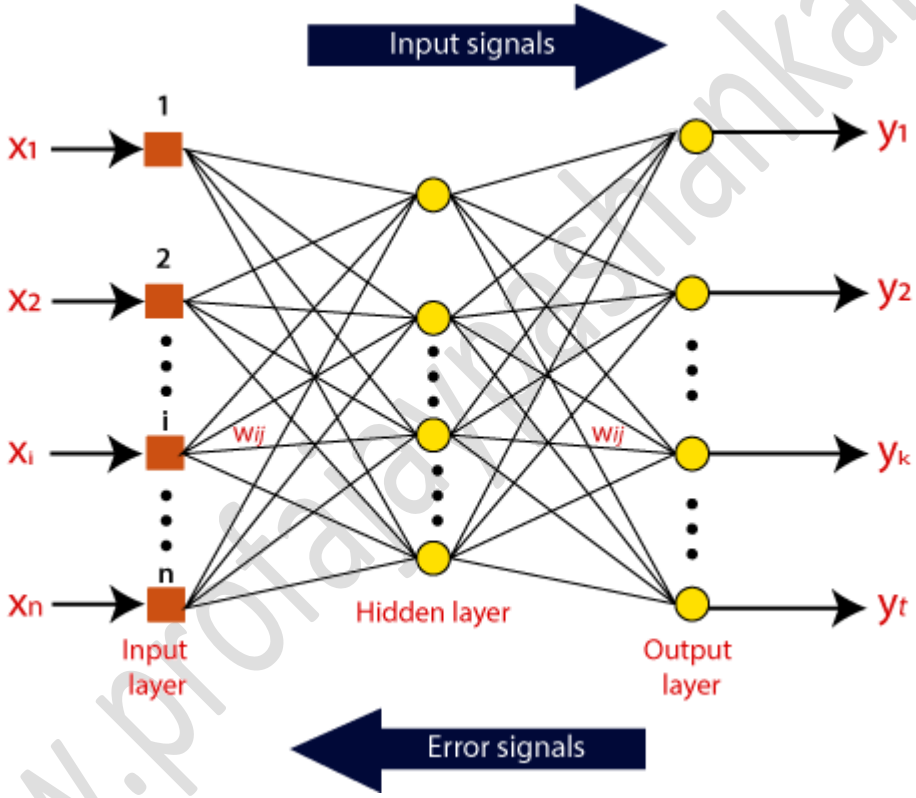
The duration of the network is unknown:

The network is reduced to a specific value of the error, and this value does not give us optimum results.

Science artificial neural networks that have steeped into the world in the mid-20th century are exponentially developing. In the present time, we have investigated the pros of artificial neural networks and the issues encountered in the course of their utilization. It should not be overlooked that the cons of ANN networks, which are a flourishing science branch, are eliminated individually, and their pros are increasing day by day. It means that artificial neural networks will turn into an irreplaceable part of our lives progressively important.

How do artificial neural networks work?

Artificial Neural Network can be best represented as a weighted directed graph, where the artificial neurons form the nodes. The association between the neurons outputs and neuron inputs can be viewed as the directed edges with weights. The Artificial Neural Network receives the input signal from the external source in the form of a pattern and image in the form of a vector. These inputs are then mathematically assigned by the notations x(n) for every n number of inputs.



Afterward, each of the input is multiplied by its corresponding weights (these weights are the details utilized by the artificial neural networks to solve a specific problem). In general terms, these weights normally represent the strength of the interconnection between neurons inside the artificial neural network. All the weighted inputs are summarized inside the computing unit.

If the weighted sum is equal to zero, then bias is added to make the output non-zero or something else to scale up to the system's response. Bias has the same input, and weight equals to 1. Here the total of weighted inputs can be in the range of 0 to positive infinity. Here, to keep the response in the limits of the desired value, a certain maximum value is benchmarked, and the total of weighted inputs is passed through the activation function.

The activation function refers to the set of transfer functions used to achieve the desired output. There is a different kind of the activation function, but primarily either linear or non-linear sets of functions. Some of the commonly used sets of activation functions are the Binary, linear, and Tan hyperbolic sigmoidal activation functions. Let us take a look at each of them in details:

Binary:

In binary activation function, the output is either a one or a 0. Here, to accomplish this, there is a threshold value set up. If the net weighted input of neurons is more than 1, then the final output of the activation function is returned as one or else the output is returned as 0.

Sigmoidal Hyperbolic:

The Sigmoidal Hyperbola function is generally seen as an "**S**" shaped curve. Here the tan hyperbolic function is used to approximate output from the actual net input. The function is defined as:

F(x) = (1/1 + exp(-????x))

Where ???? is considered the Steepness parameter.

Types of Artificial Neural Network:

There are various types of Artificial Neural Networks (ANN) depending upon the human brain neuron and network functions, an artificial neural network similarly performs tasks. The majority of the artificial neural networks will have some similarities with a more complex biological partner and are very effective at their expected tasks. For example, segmentation or classification.

Feedback ANN:

In this type of ANN, the output returns into the network to accomplish the best-evolved results internally. As per the **University of Massachusetts**, Lowell Centre for Atmospheric Research. The feedback networks feed information back into itself and are well suited to solve optimization issues. The Internal system error corrections utilize feedback ANNs.

Feed-Forward ANN:

A feed-forward network is a basic neural network comprising of an input layer, an output layer, and at least one layer of a neuron. Through assessment of its output by reviewing its input, the intensity of the network can be noticed based on group behavior of the associated neurons, and the output is decided. The primary advantage of this network is that it figures out how to evaluate and recognize input patterns.

Basic Models of Artificial Neural Network

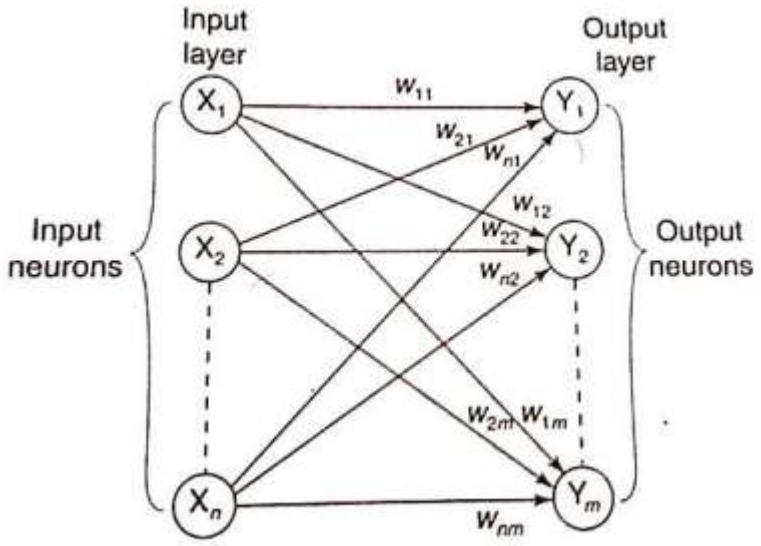
The models of ANN are specified by the three basic entities

1. The model's synaptic interconnections
2. The learning rules adopted for updating and adjusting the connection weights
3. The activation functions

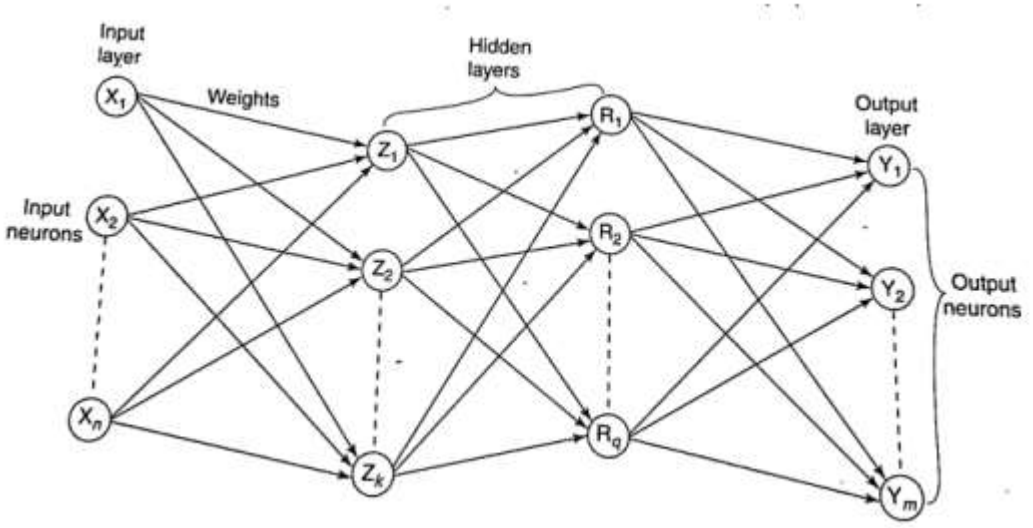
The model's synaptic interconnections ANN consists of a set of highly interconnected neurons connected through weights to the other processing elements or to itself. The arrangement of these processing elements and the geometry of their interconnections are important for ANN. The arrangement of neurons to form layers and the connection pattern formed within and between layers is called the network architecture. There are five basic neuron connection architectures.

1. Single-layer feed-forward network
2. Multilayer feed-forward network
3. Single node with its own feedback
4. Single-layer recurrent network
5. Multi-layer recurrent network

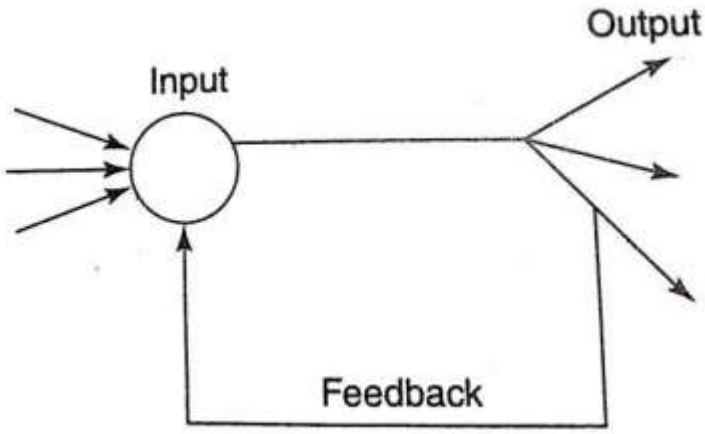
1. Single-layer feed-forward network It consists of a single layer of network where the inputs are directly connected to the output, one per node with a series of various weights.



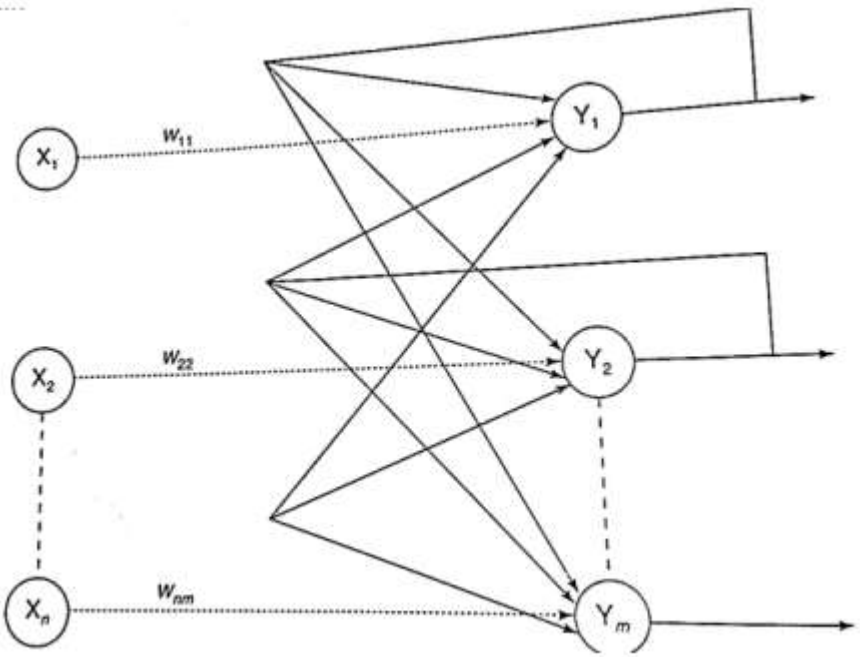
2. Multi-layer feed-forward network It consists of multi layers where along with the input and output layers, there are hidden layers. There can be zero to many hidden layers. The hidden layer is usually internal to the network and has no direct contact with the environment.



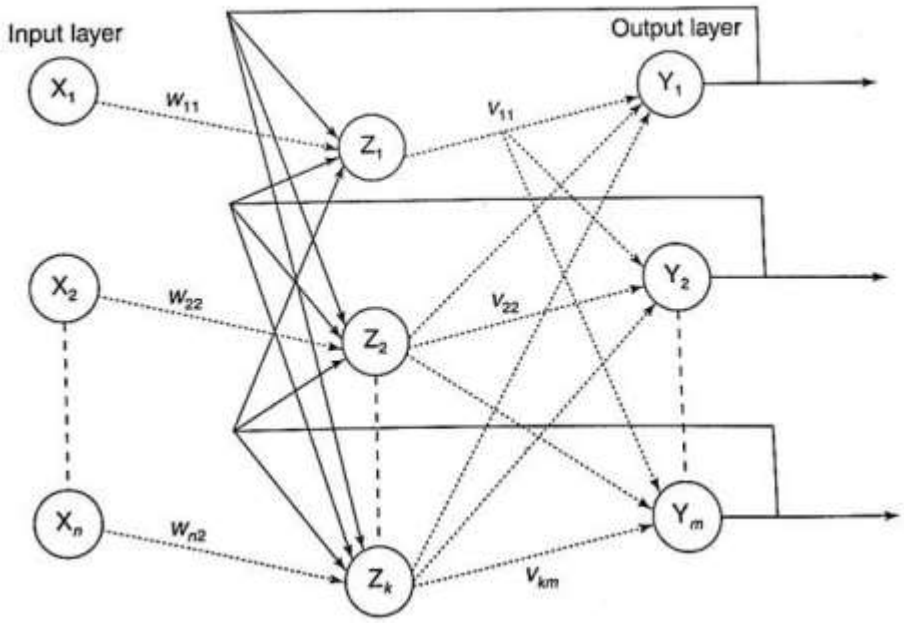
3. Single node with own feedback The simplest neural network architecture giving feedback to itself with a single neuron.



4. Single-layer recurrent network A single-layer network with a feedback directed back to itself or to other processing element or both.



5. Multilayer recurrent network A recurrent network has at least a feedback in place. The processing elements output can be directed back to the nodes in the previous layer.



What are the design issues in an Artificial Neural Network?

- An artificial neural network is a system based on the functions of biological neural networks.
- It is a simulation of a biological neural system.
- The feature of artificial neural networks is that there are several structures, which required several methods of algorithms, but regardless of being a complex system, a neural network is easy.
- These networks are between the specific signal-processing sciences in the director's toolbox.
- The area is hugely interdisciplinary, but this approach will restrict the view to the engineering perspective.
- In engineering, neural networks produce two essential functions as pattern classifiers and as non-linear adaptive filters. An artificial neural network is dynamic, it provides a non-linear system that learns to execute a function (an input/output map) from data.
- Adaptive represents that the system parameters are changed during operation, frequently called the training phase.
- After the training phase, the artificial neural network parameters are fixed and the system is start to solve the problem at hand (the testing phase).
- The artificial neural network is produced with a systematic step-by-step procedure to improve a performance test or to follow some definite internal constraint, which is usually described as the learning rule.

There are the following design issues that must be considered which are as follows –

- The several nodes in the input layer must be decided. It can be created an input node for each mathematical or binary input variable.
- If the input variable is categorical, it can create one node for each categorical value or encrypt the k-ary variable using $[\log_2 k]$ input nodes.
- The multiple nodes in the output layer must be created.
- For a two-class problem, it is adequate to need a single output node. For a k-class problem, there are k output nodes.
- The network topology such as the number of hidden layers and hidden nodes, and feed-forward or recurrent network structure should be selected.
- The target function description is based on the weights of the connection, the multiple hidden nodes and hidden layers, biases in the nodes, and the type of activation function.
- It is discovering the right topology is not a simple task.
- One method is to start from a fully connected network with an adequately huge number of nodes and hidden layers, and then repeat the model-building structure with a smaller number of nodes.
- This method can be moderate.
- Alternatively, instead of repeating the model-building structure, it can delete several nodes and repeat the model evaluation process to choose the right model complexity.
- The weights and biases should be initialized. Random assignments are generally adequate.
- Training instances with missing values must be deleted or restored with the most likely values.

CHAPTER VII: EVOLUTIONARY COMPUTATION

Topics covered:

: Soft computing, genetic algorithms, genetic programming concepts, evolutionary programming, swarm intelligence, ant colony paradigm, particle swarm optimization and applications of evolutionary algorithms

---

Soft computing:

Soft computing is defined as a group of computational techniques based on artificial intelligence (human like decision) and natural selection that provides quick and cost effective solution to very complex problems for which analytical (hard computing) formulations do not exist. The term soft computing was coined by Zadeh [Zadeh, 1992]. Soft computing aims at finding precise approximation, which gives a robust, computationally efficient and cost effective solution saving the computational time. Most of these techniques are basically enthused on biological inspired phenomena and societal behavioural patterns. The advent of soft computing into the computing world was marked by research in machine learning, probabilistic reasoning, artificial neural networks (ANN), fuzzy logic [Jang *et al*., 1997] and genetic algorithm (GA). Today, the purview of soft computing has been extended to include swarm intelligence and foraging behaviours of biological populations in algorithms like the particle swarm optimization (PSO) and bacterial foraging algorithm (BFO) [Holland, 1975; Kennedy and Eberhart, 1995; Passino, 2002].

Soft computing methods are associated with certain distinctive advantages. These include the following:

1. • Since Soft computing methods do not call for wide-ranging mathematical formulation pertaining to the problem, the need for explicit knowledge in a particular domain can be reduced.
2. • These tools can handle multiple variables simultaneously.
3. • For optimization problems, the solutions can be prevented from falling into local minima by using global optimization strategies.
4. • These techniques are mostly cost effective.
5. • Dependency on expensive traditional simulations packages can be reduced to some degree by efficient hybridization of soft computing methods.
6. • These methods are generally adaptive in nature and are scalable.

Of late, soft computing techniques have attracted recognition amongst researchers of various branches of engineering in order to arrive at solutions to problem statements [Patnaik and Mishra, 2000; Patnaik *et al*., 2005; Samii, 2006; Choudhury *et al*., 2012]. The sturdiness of the above techniques has been well tested pertaining to various problems encountered in every sphere of engineering. Indeed, the last decade has seen the implementation of soft computing in microwave applications. This chapter gives a glimpse of the various soft computing techniques that are widely used in the field of electromagnetics.

What is soft computing?

Soft computing is the reverse of hard (conventional) computing. It refers to a group of computational techniques that are based on artificial intelligence (AI) and natural selection. It provides cost-effective solutions to the complex real-life problems for which hard computing solution does not exist.

Zadeh coined the term of soft computing in 1992. The objective of soft computing is to provide precise approximation and quick solutions for complex real-life problems.

In simple terms, you can understand soft computing - an emerging approach that gives the amazing ability of the human mind. It can map a human mind and the human mind is a role model for soft computing.

Note: Basically, soft computing is different from traditional/conventional computing and it deals with approximation models.

Some characteristics of Soft computing

- Soft computing provides an approximate but precise solution for real-life problems.
- The algorithms of soft computing are adaptive, so the current process is not affected by any kind of change in the environment.
- The concept of soft computing is based on **learning from experimental data**. It means that soft computing does not require any mathematical model to solve the problem.
- Soft computing helps users to solve real-world problems by providing approximate results that conventional and analytical models cannot solve.
- It is based on Fuzzy logic, genetic algorithms, machine learning, ANN, and expert systems.

Example

Soft computing deals with the approximation model. Yoi will understand with the help of examples of how it deals with the approximation model.

Let's consider a problem that actually does not have any solution via traditional computing, but soft computing gives the approximate solution.

string1 = "xyz" and string2 = "xyw"

1. Problem 1
2. Are string1 and string2 same?
3. Solution
4. No, the solution is simply No. It does not require any algorithm to analyze this.

Let's modify the problem a bit.

1. Problem 2
2. How much string1 and string2 are same?
3. Solution
4. Through conventional programming, either the answer is Yes or No. But these strings might be 80% similar according to soft computing.

You have noticed that soft computing gave us the approximate solution.

Applications of soft computing

There are several applications of soft computing where it is used. Some of them are listed below:

- It is widely used in **gaming products like Poker and Checker**.
- In kitchen appliances, such as **Microwave and Rice cooker**.
- In most used home appliances - **Washing Machine, Heater, Refrigerator, and AC** as well.
- Apart from all these usages, it is also used in **Robotics work** (Emotional per Robot form).
- **Image processing and Data compression** are also popular applications of soft computing.
- Used for handwriting recognition.

As we already said that, soft computing provides the solution to real-time problems and here you can see that. Besides these applications, there are many other applications of soft computing.

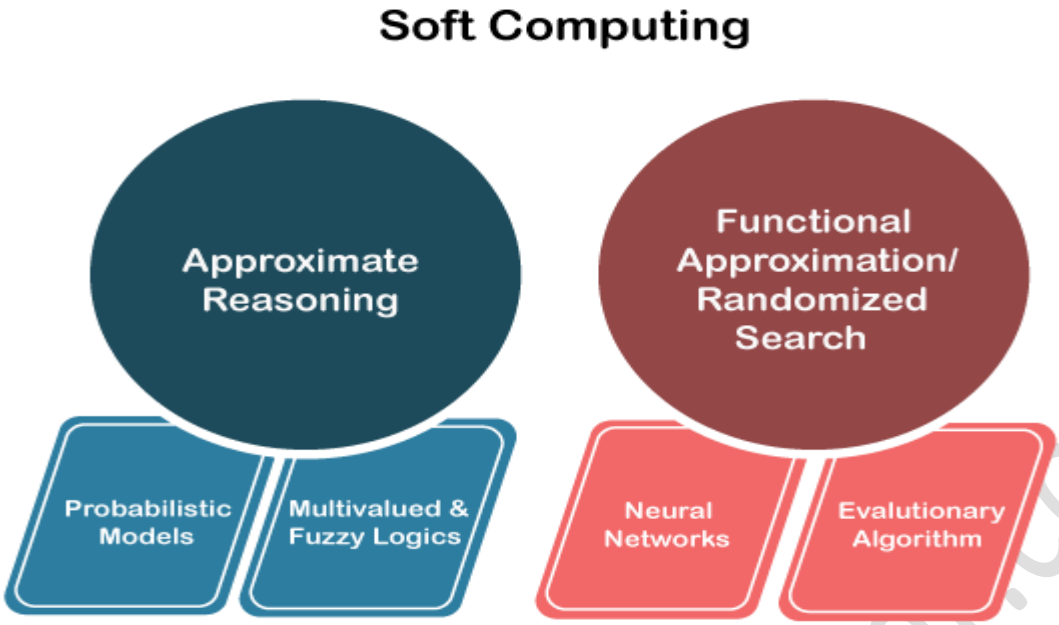
Need of soft computing

Sometimes, conventional computing or analytical models does not provide a solution to some real-world problems. In that case, we require other technique like soft computing to obtain an approximate solution.

- Hard computing is used for solving mathematical problems that need a precise answer. It fails to provide solutions for some real-life problems. Thereby for real-life problems whose precise solution does not exist, soft computing helps.
- When conventional mathematical and analytical models fail, soft computing helps, e.g., You can map even the human mind using soft computing.
- Analytical models can be used for solving mathematical problems and valid for ideal cases. But the real-world problems do not have an ideal case; these exist in a non-ideal environment.
- Soft computing is not only limited to theory; it also gives insights into real-life problems.
- Like all the above reasons, Soft computing helps to map the human mind, which cannot be possible with conventional mathematical and analytical models.

Elements of soft computing

Soft computing is viewed as a foundation component for an emerging field of conceptual intelligence. Fuzzy Logic (FL), Machine Learning (ML), Neural Network (NN), Probabilistic Reasoning (PR), and Evolutionary Computation (EC) are the supplements of soft computing. Also, these are techniques used by soft computing to resolve any complex problem.



Any problems can be resolved effectively using these components. Following are three types of techniques used by soft computing:

- Fuzzy Logic
- Artificial Neural Network (ANN)
- Genetic Algorithms

Fuzzy Logic (FL)

Fuzzy logic is nothing but mathematical logic which tries to solve problems with an open and imprecise spectrum of data. It makes it easy to obtain an array of precise conclusions.

Fuzzy logic is basically designed to achieve the best possible solution to complex problems from all the available information and input data. Fuzzy logics are considered as the best solution finders.

Neural Network (ANN)

Neural networks were developed in the 1950s, which helped soft computing to solve real-world problems, which a computer cannot do itself. We all know that a human brain can easily describe real-world conditions, but a computer cannot.

An artificial neural network (ANN) emulates a network of neurons that makes a human brain (means a machine that can think like a human mind). Thereby the computer or a machine can learn things so that they can take decisions like the human brain.

Artificial Neural Networks (ANN) are mutually connected with brain cells and created using regular computing programming. It is like as the human neural system.

Genetic Algorithms (GA)

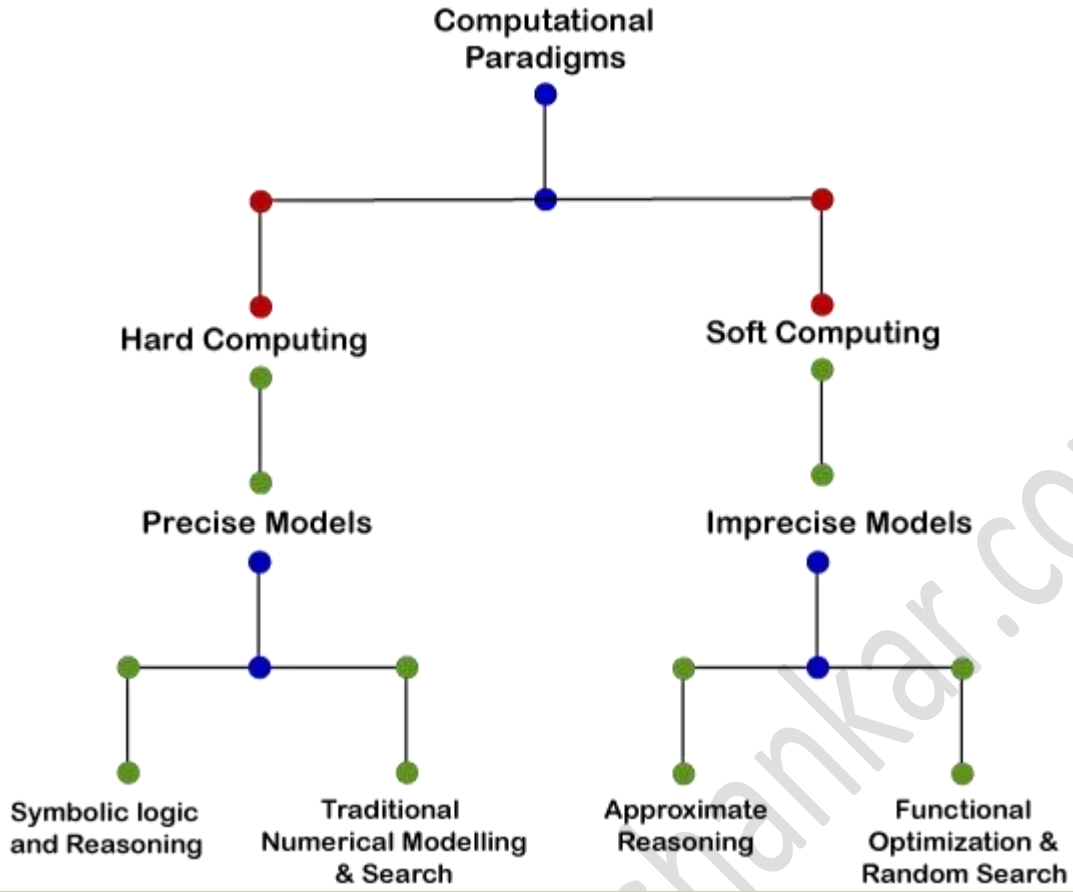
Genetic algorithm is almost based on nature and take all inspirations from it. There is no genetic algorithm that is based on search-based algorithms, which find its roots in natural selection and the concept of genetics.

In addition, a genetic algorithm is a subset of a large branch of computation.

Soft computing vs hard computing

Hard computing uses existing mathematical algorithms to solve certain problems. It provides a precise and exact solution of the problem. Any numerical problem is an example of hard computing.

On the other hand, soft computing is a different approach than hard computing. In soft computing, we compute solutions to the existing complex problems. The result calculated or provided by soft computing are also not precise. They are imprecise and fuzzy in nature.



Parameters	Soft Computing	Hard Computing
Computation time	Takes less computation time.	Takes more computation time.
Dependency	It depends on approximation and dispositional.	It is mainly based on binary logic and numerical systems.
Computation type	Parallel computation	Sequential computation
Result/Output	Approximate result	Exact and precise result
Example	Neural Networks, such as Madaline, Adaline, Art Networks.	Any numerical problem or traditional methods of solving using personal computers.

Genetic Algorithms

Genetic Algorithms - Introduction

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of **Genetics and Natural Selection**. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.

Introduction to Optimization

Optimization is the process of **making something better**. In any process, we have a set of inputs and a set of outputs as shown in the following figure.

Optimization refers to finding the values of inputs in such a way that we get the "best" output values. The definition of "best" varies from problem to problem, but in mathematical terms, it refers to maximizing or minimizing one or more objective functions, by varying the input parameters.

The set of all possible solutions or values which the inputs can take make up the search space. In this search space, lies a point or a set of points which gives the optimal solution. The aim of optimization is to find that point or set of points in the search space.

What are Genetic Algorithms?

Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as **Evolutionary Computation**.

GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.

In GAs, we have a **pool or a population of possible solutions** to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more "fitter" individuals. This is in line with the Darwinian Theory of "Survival of the Fittest".

In this way we keep "evolving" better individuals or solutions over generations, till we reach a stopping criterion.

Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.

Advantages of GAs

GAs have various advantages which have made them immensely popular. These include –

- Does not require any derivative information (which may not be available for many real-world problems).
- Is faster and more efficient as compared to the traditional methods.
- Has very good parallel capabilities.
- Optimizes both continuous and discrete functions and also multi-objective problems.
- Provides a list of "good" solutions and not just a single solution.
- Always gets an answer to the problem, which gets better over the time.
- Useful when the search space is very large and there are a large number of parameters involved.

Limitations of GAs

Like any technique, GAs also suffer from a few limitations. These include –

- GAs are not suited for all problems, especially problems which are simple and for which derivative information is available.
- Fitness value is calculated repeatedly which might be computationally expensive for some problems.
- Being stochastic, there are no guarantees on the optimality or the quality of the solution.
- If not implemented properly, the GA may not converge to the optimal solution.

GA – Motivation

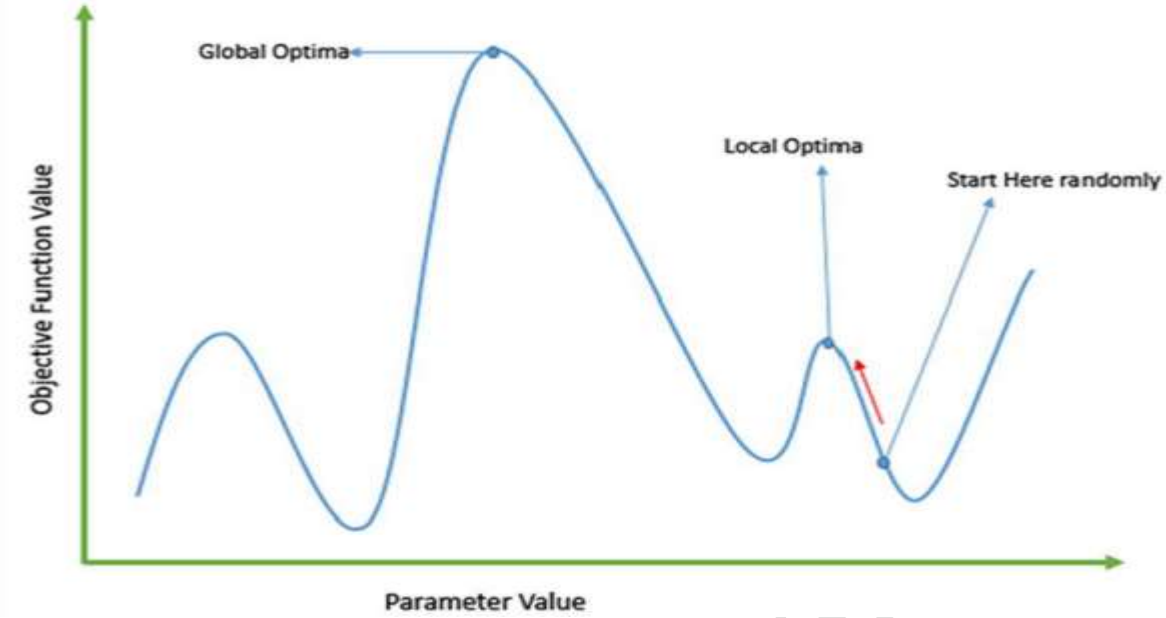
Genetic Algorithms have the ability to deliver a "good-enough" solution "fast-enough". This makes genetic algorithms attractive for use in solving optimization problems. The reasons why GAs are needed are as follows –

Solving Difficult Problems

In computer science, there is a large set of problems, which are **NP-Hard**. What this essentially means is that, even the most powerful computing systems take a very long time (even years!) to solve that problem. In such a scenario, GAs prove to be an efficient tool to provide **usable near-optimal solutions** in a short amount of time.

Failure of Gradient Based Methods

Traditional calculus based methods work by starting at a random point and by moving in the direction of the gradient, till we reach the top of the hill. This technique is efficient and works very well for single-peaked objective functions like the cost function in linear regression. But, in most real-world situations, we have a very complex problem called as landscapes, which are made of many peaks and many valleys, which causes such methods to fail, as they suffer from an inherent tendency of getting stuck at the local optima as shown in the following figure.



Getting a Good Solution Fast

Some difficult problems like the Travelling Salesperson Problem (TSP), have real-world applications like path finding and VLSI Design. Now imagine that you are using your GPS Navigation system, and it takes a few minutes (or even a few hours) to compute the "optimal" path from the source to destination. Delay in such real world applications is not acceptable and therefore a "good-enough" solution, which is delivered "fast" is what is required.

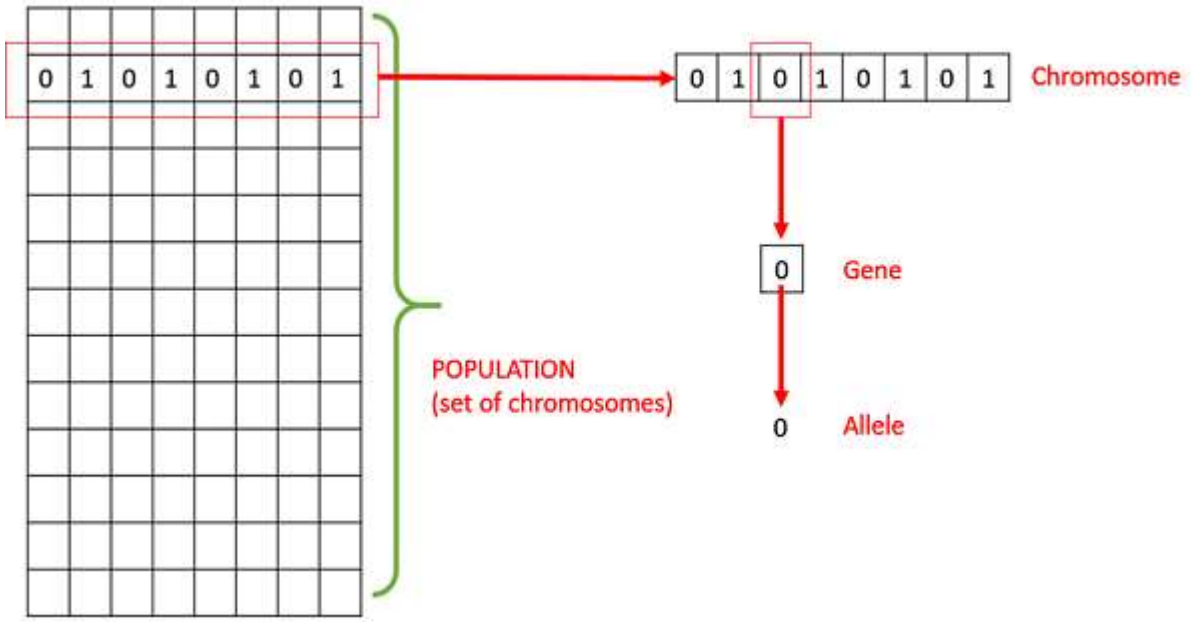
Genetic Algorithms - Fundamentals

This section introduces the basic terminology required to understand GAs. Also, a generic structure of GAs is presented in both **pseudo-code and graphical forms**. The reader is advised to properly understand all the concepts introduced in this section and keep them in mind when reading other sections of this tutorial as well.

Basic Terminology

Before beginning a discussion on Genetic Algorithms, it is essential to be familiar with some basic terminology which will be used throughout this tutorial.

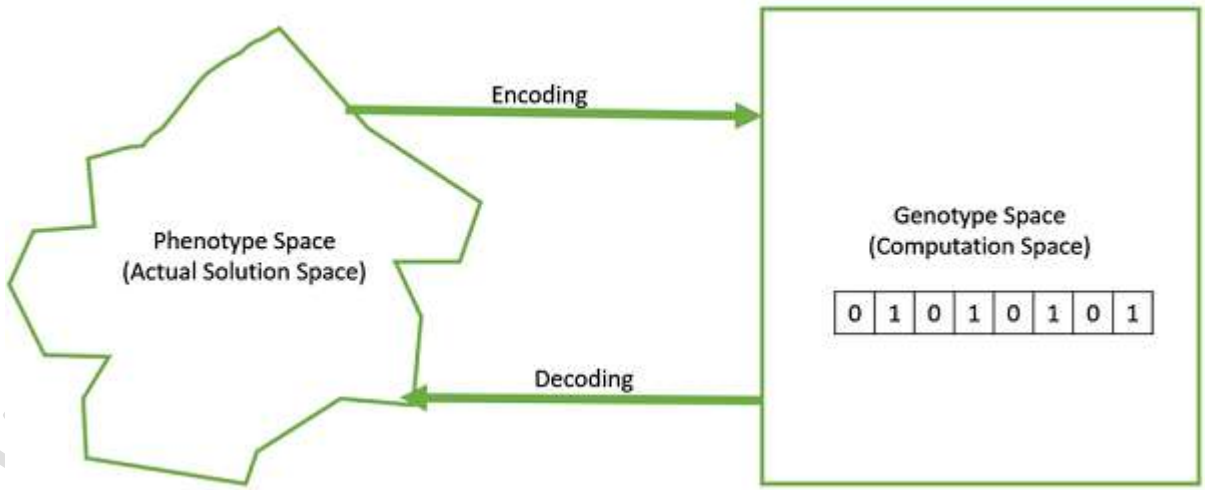
- **Population** − It is a subset of all the possible (encoded) solutions to the given problem. The population for a GA is analogous to the population for human beings except that instead of human beings, we have Candidate Solutions representing human beings.
- **Chromosomes** − A chromosome is one such solution to the given problem.
- **Gene** − A gene is one element position of a chromosome.
- **Allele** − It is the value a gene takes for a particular chromosome.



- **Genotype** – Genotype is the population in the computation space. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system.
- **Phenotype** – Phenotype is the population in the actual real world solution space in which solutions are represented in a way they are represented in real world situations.
- **Decoding and Encoding** – For simple problems, the **phenotype and genotype** spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. Decoding is a process of transforming a solution from the genotype to the phenotype space, while encoding is a process of transforming from the phenotype to genotype space. Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.

For example, consider the 0/1 Knapsack Problem. The Phenotype space consists of solutions which just contain the item numbers of the items to be picked.

However, in the genotype space it can be represented as a binary string of length n (where n is the number of items). A **0 at position x** represents that **xth** item is picked while a 1 represents the reverse. This is a case where genotype and phenotype spaces are different.



- **Fitness Function** – A fitness function simply defined is a function which takes the solution as input and produces the suitability of the solution as the output. In some cases, the fitness function and the objective function may be the same, while in others it might be different based on the problem.
- **Genetic Operators** – These alter the genetic composition of the offspring. These include crossover, mutation, selection, etc.

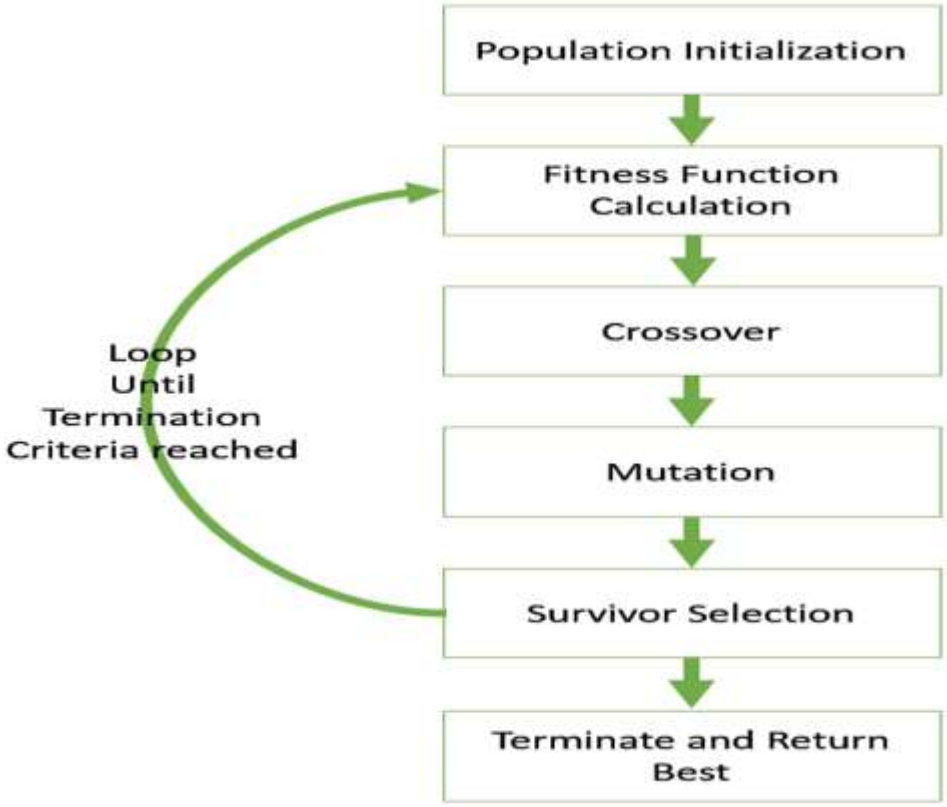
Basic Structure

The basic structure of a GA is as follows –

We start with an initial population (which may be generated at random or seeded by other heuristics), select parents from this population for mating. Apply crossover and mutation operators on the parents to generate new off-springs. And finally these off-springs replace the existing individuals in the population and

the process repeats. In this way genetic algorithms actually try to mimic the human evolution to some extent.

Each of the following steps are covered as a separate chapter later in this tutorial.



A generalized pseudo-code for a GA is explained in the following program –

```
GA()
  initialize population
  find fitness of population
  while (termination criteria is reached) do
    parent selection
    crossover with probability pc
    mutation with probability pm
    decode and fitness calculation
    survivor selection
    find best
  return best
```

Genotype Representation

One of the most important decisions to make while implementing a genetic algorithm is deciding the representation that we will use to represent our solutions. It has been observed that improper representation can lead to poor performance of the GA.

Therefore, choosing a proper representation, having a proper definition of the mappings between the phenotype and genotype spaces is essential for the success of a GA.

In this section, we present some of the most commonly used representations for genetic algorithms. However, representation is highly problem specific and the reader might find that another representation or a mix of the representations mentioned here might suit his/her problem better.

Binary Representation

This is one of the simplest and most widely used representation in GAs. In this type of representation the genotype consists of bit strings.

For some problems when the solution space consists of Boolean decision variables – yes or no, the binary representation is natural. Take for example the 0/1 Knapsack Problem. If there are n items, we can represent a solution by a binary string of n elements, where the x^{th} element tells whether the item x is picked (1) or not (0).

For other problems, specifically those dealing with numbers, we can represent the numbers with their binary representation. The problem with this kind of encoding is that different bits have different significance and therefore mutation and crossover operators can have undesired consequences. This can be

resolved to some extent by using **Gray Coding,** as a change in one bit does not have a massive effect on the solution.

Real Valued Representation

For problems where we want to define the genes using continuous rather than discrete variables, the real valued representation is the most natural. The precision of these real valued or floating point numbers is however limited to the computer.

0.5	0.2	0.6	0.8	0.7	0.4	0.3	0.2	0.1	0.9

Integer Representation

For discrete valued genes, we cannot always limit the solution space to binary 'yes' or 'no'. For example, if we want to encode the four distances – North, South, East and West, we can encode them as **{0,1,2,3}**. In such cases, integer representation is desirable.

1	2	3	4	3	2	4	1	2	1

Permutation Representation

In many problems, the solution is represented by an order of elements. In such cases permutation representation is the most suited.

A classic example of this representation is the travelling salesman problem (TSP). In this the salesman has to take a tour of all the cities, visiting each city exactly once and come back to the starting city. The total distance of the tour has to be minimized. The solution to this TSP is naturally an ordering or permutation of all the cities and therefore using a permutation representation makes sense for this problem.

1	5	9	8	7	4	2	3	6	0

Genetic Algorithms - Population

Population is a subset of solutions in the current generation. It can also be defined as a set of chromosomes. There are several things to be kept in mind when dealing with GA population –

- The diversity of the population should be maintained otherwise it might lead to premature convergence.
- The population size should not be kept very large as it can cause a GA to slow down, while a smaller population might not be enough for a good mating pool. Therefore, an optimal population size needs to be decided by trial and error.

The population is usually defined as a two dimensional array of – **size population, size x, chromosome size**.

Population Initialization

There are two primary methods to initialize a population in a GA. They are –

- **Random Initialization** – Populate the initial population with completely random solutions.
- **Heuristic initialization** – Populate the initial population using a known heuristic for the problem.

It has been observed that the entire population should not be initialized using a heuristic, as it can result in the population having similar solutions and very little diversity. It has been experimentally observed that the random solutions are the ones to drive the population to optimality. Therefore, with heuristic initialization, we just seed the population with a couple of good solutions, filling up the rest with random solutions rather than filling the entire population with heuristic based solutions.

It has also been observed that heuristic initialization in some cases, only effects the initial fitness of the population, but in the end, it is the diversity of the solutions which lead to optimality.

Population Models

There are two population models widely in use –

Steady State

In steady state GA, we generate one or two off-springs in each iteration and they replace one or two individuals from the population. A steady state GA is also known as **Incremental GA**.

Generational

In a generational model, we generate 'n' off-springs, where n is the population size, and the entire population is replaced by the new one at the end of the iteration.

Genetic Algorithms - Fitness Function

The fitness function simply defined is a function which takes a **candidate solution to the problem as input and produces as output** how "fit" our how "good" the solution is with respect to the problem in consideration.

Calculation of fitness value is done repeatedly in a GA and therefore it should be sufficiently fast. A slow computation of the fitness value can adversely affect a GA and make it exceptionally slow.

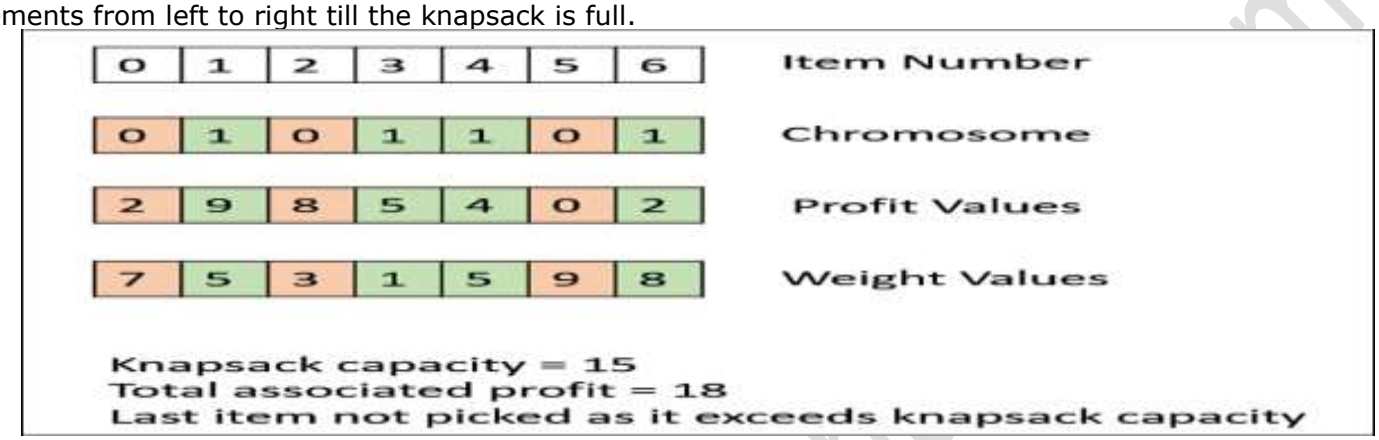
In most cases the fitness function and the objective function are the same as the objective is to either maximize or minimize the given objective function. However, for more complex problems with multiple objectives and constraints, an **Algorithm Designer** might choose to have a different fitness function.

A fitness function should possess the following characteristics –

- The fitness function should be sufficiently fast to compute.
- It must quantitatively measure how fit a given solution is or how fit individuals can be produced from the given solution.

In some cases, calculating the fitness function directly might not be possible due to the inherent complexities of the problem at hand. In such cases, we do fitness approximation to suit our needs.

The following image shows the fitness calculation for a solution of the 0/1 Knapsack. It is a simple fitness function which just sums the profit values of the items being picked (which have a 1), scanning the elements from left to right till the knapsack is full.



Genetic Algorithms - Parent Selection

Parent Selection is the process of selecting parents which mate and recombine to create off-springs for the next generation. Parent selection is very crucial to the convergence rate of the GA as good parents drive individuals to a better and fitter solutions.

However, care should be taken to prevent one extremely fit solution from taking over the entire population in a few generations, as this leads to the solutions being close to one another in the solution space thereby leading to a loss of diversity. **Maintaining good diversity** in the population is extremely crucial for the success of a GA. This taking up of the entire population by one extremely fit solution is known as **premature convergence** and is an undesirable condition in a GA.

Fitness Proportionate Selection

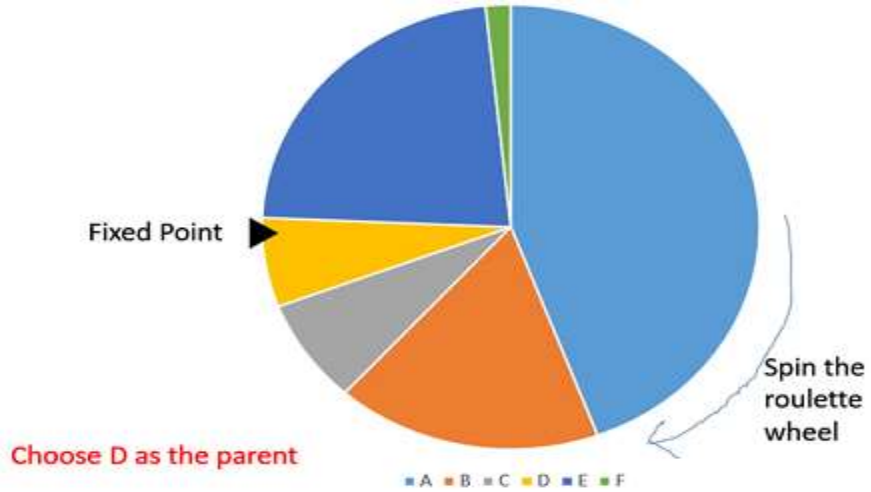
Fitness Proportionate Selection is one of the most popular ways of parent selection. In this every individual can become a parent with a probability which is proportional to its fitness. Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation. Therefore, such a selection strategy applies a selection pressure to the more fit individuals in the population, evolving better individuals over time.

Consider a circular wheel. The wheel is divided into **n pies**, where n is the number of individuals in the population. Each individual gets a portion of the circle which is proportional to its fitness value.

Two implementations of fitness proportionate selection are possible –

Roulette Wheel Selection

In a roulette wheel selection, the circular wheel is divided as described before. A fixed point is chosen on the wheel circumference as shown and the wheel is rotated. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

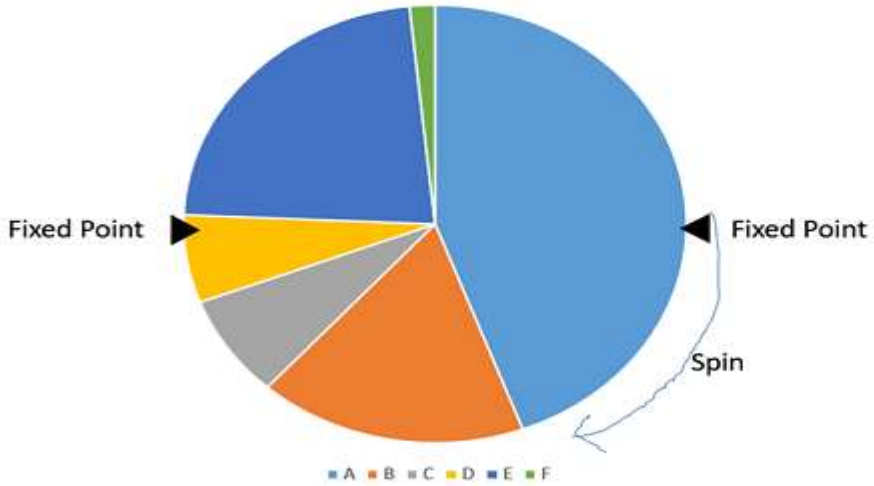
It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. Therefore, the probability of choosing an individual depends directly on its fitness.

Implementation wise, we use the following steps –

- Calculate S = the sum of a finesses.
- Generate a random number between 0 and S.
- Starting from the top of the population, keep adding the finesses to the partial sum P, till P<S.
- The individual for which P exceeds S is the chosen individual.

Stochastic Universal Sampling (SUS)

Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the following image. Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once.

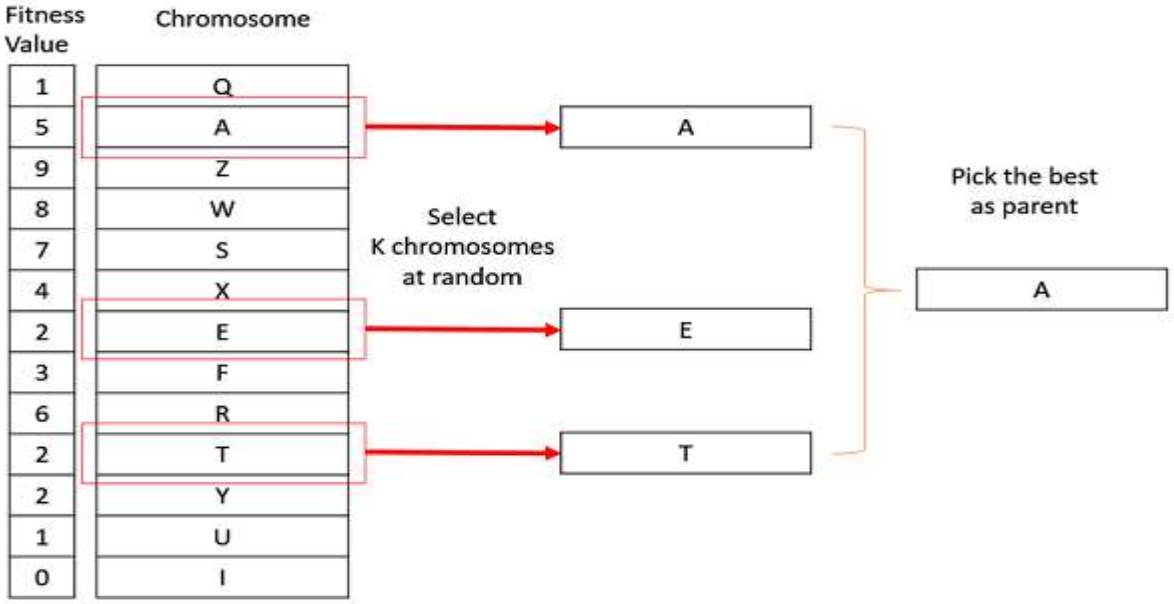


Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

It is to be noted that fitness proportionate selection methods don't work for cases where the fitness can take a negative value.

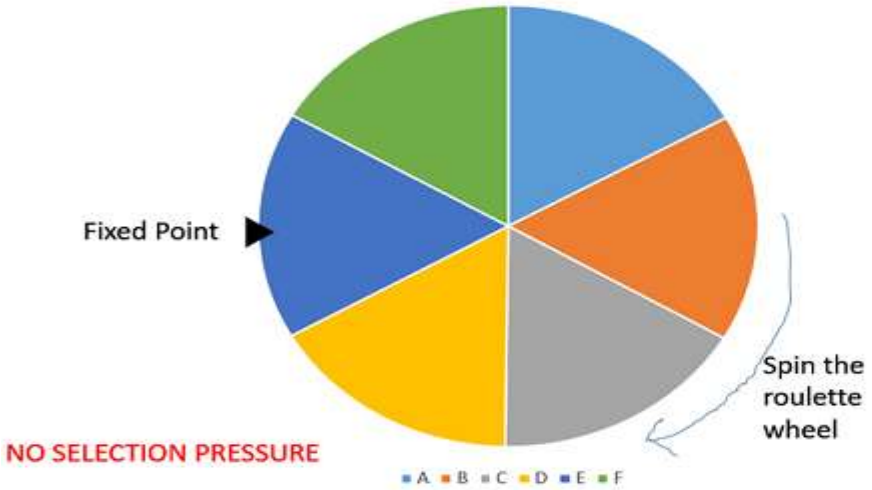
Tournament Selection

In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.



Rank Selection

Rank Selection also works with negative fitness values and is mostly used when the individuals in the population have very close fitness values (this happens usually at the end of the run). This leads to each individual having an almost equal share of the pie (like in case of fitness proportionate selection) as shown in the following image and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent. This in turn leads to a loss in the selection pressure towards fitter individuals, making the GA to make poor parent selections in such situations.



Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

In this, we remove the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. The selection of the parents depends on the rank of each individual and not the fitness. The higher ranked individuals are preferred more than the lower ranked ones.

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

Random Selection

In this strategy we randomly select parents from the existing population. There is no selection pressure towards fitter individuals and therefore this strategy is usually avoided.

Genetic Algorithms - Crossover

In this chapter, we will discuss about what a Crossover Operator is along with its other modules, their uses and benefits.

Introduction to Crossover

The crossover operator is analogous to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents. Crossover is usually applied in a GA with a high probability – p_c .

Crossover Operators

In this section we will discuss some of the most popularly used crossover operators. It is to be noted that these crossover operators are very generic and the GA Designer might choose to implement a problem-specific crossover operator as well.

One Point Crossover

In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.

Multi Point Crossover

Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.

Uniform Crossover

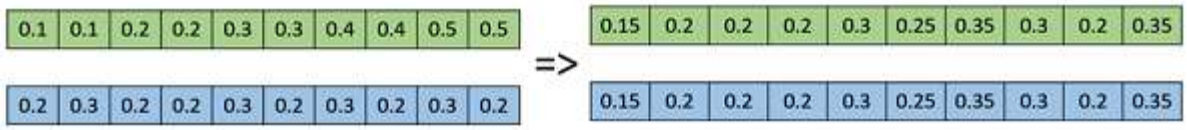
In a uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring. We can also bias the coin to one parent, to have more genetic material in the child from that parent.

Whole Arithmetic Recombination

This is commonly used for integer representations and works by taking the weighted average of the two parents by using the following formulae –

- Child1 = α.x + (1-α).y
- Child2 = α.x + (1-α).y

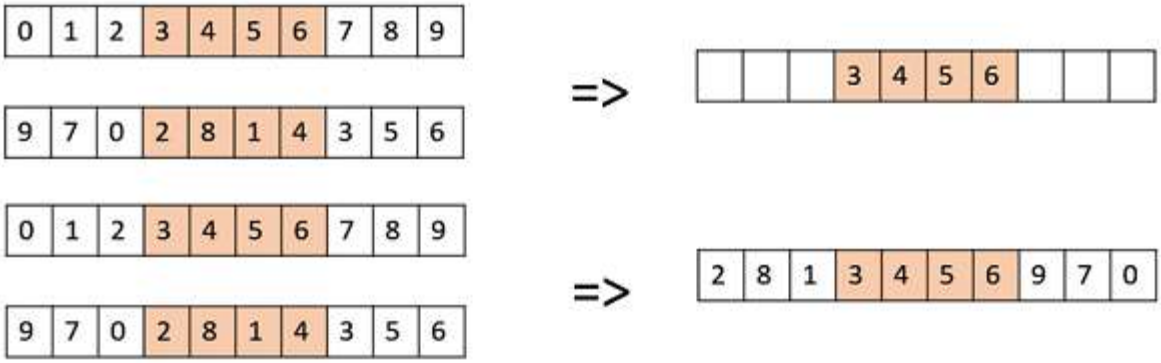
Obviously, if α = 0.5, then both the children will be identical as shown in the following image.



Davis' Order Crossover (OX1)

OX1 is used for permutation based crossovers with the intention of transmitting information about relative ordering to the off-springs. It works as follows –

- Create two random crossover points in the parent and copy the segment between them from the first parent to the first offspring.
- Now, starting from the second crossover point in the second parent, copy the remaining unused numbers from the second parent to the first child, wrapping around the list.
- Repeat for the second child with the parent's role reversed.



Repeat the same procedure to get the second child

There exist a lot of other crossovers like Partially Mapped Crossover (PMX), Order based crossover (OX2), Shuffle Crossover, Ring Crossover, etc.

Genetic Algorithms - Mutation

Introduction to Mutation

In simple terms, mutation may be defined as a small random tweak in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability – p_m. If the probability is very high, the GA gets reduced to a random search.

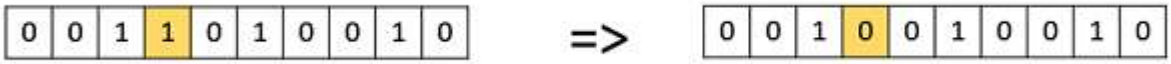
Mutation is the part of the GA which is related to the "exploration" of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is not.

Mutation Operators

In this section, we describe some of the most commonly used mutation operators. Like the crossover operators, this is not an exhaustive list and the GA designer might find a combination of these approaches or a problem-specific mutation operator more useful.

Bit Flip Mutation

In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.

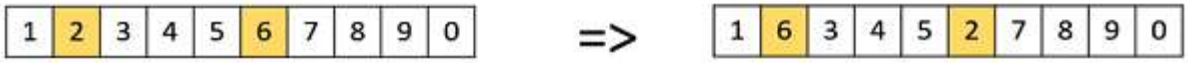


Random Resetting

Random Resetting is an extension of the bit flip for the integer representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

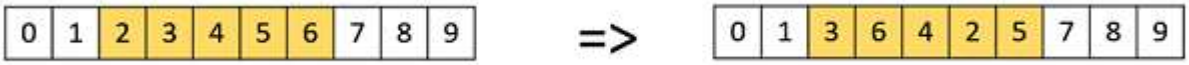
Swap Mutation

In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings.



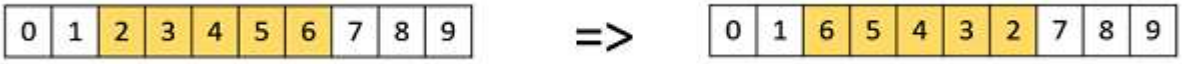
Scramble Mutation

Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.



Inversion Mutation

In inversion mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.



Genetic Algorithms - Survivor Selection

The Survivor Selection Policy determines which individuals are to be kicked out and which are to be kept in the next generation. It is crucial as it should ensure that the fitter individuals are not kicked out of the population, while at the same time diversity should be maintained in the population.

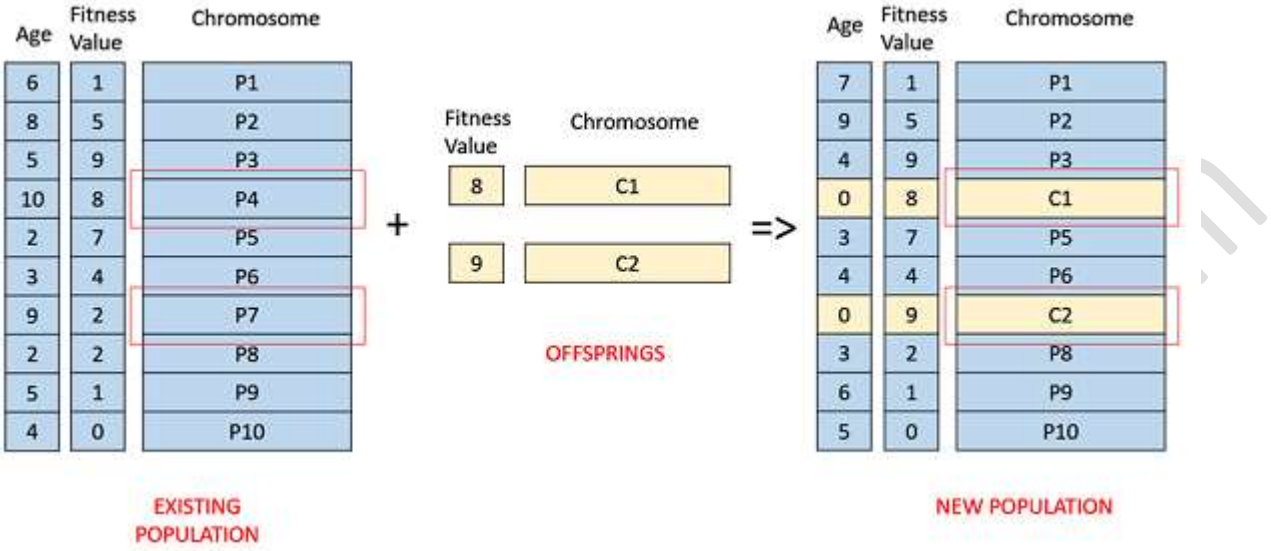
Some GAs employ **Elitism**. In simple terms, it means the current fittest member of the population is always propagated to the next generation. Therefore, under no circumstance can the fittest member of the current population be replaced.

The easiest policy is to kick random members out of the population, but such an approach frequently has convergence issues, therefore the following strategies are widely used.

Age Based Selection

In Age-Based Selection, we don't have a notion of a fitness. It is based on the premise that each individual is allowed in the population for a finite generation where it is allowed to reproduce, after that, it is kicked out of the population no matter how good its fitness is.

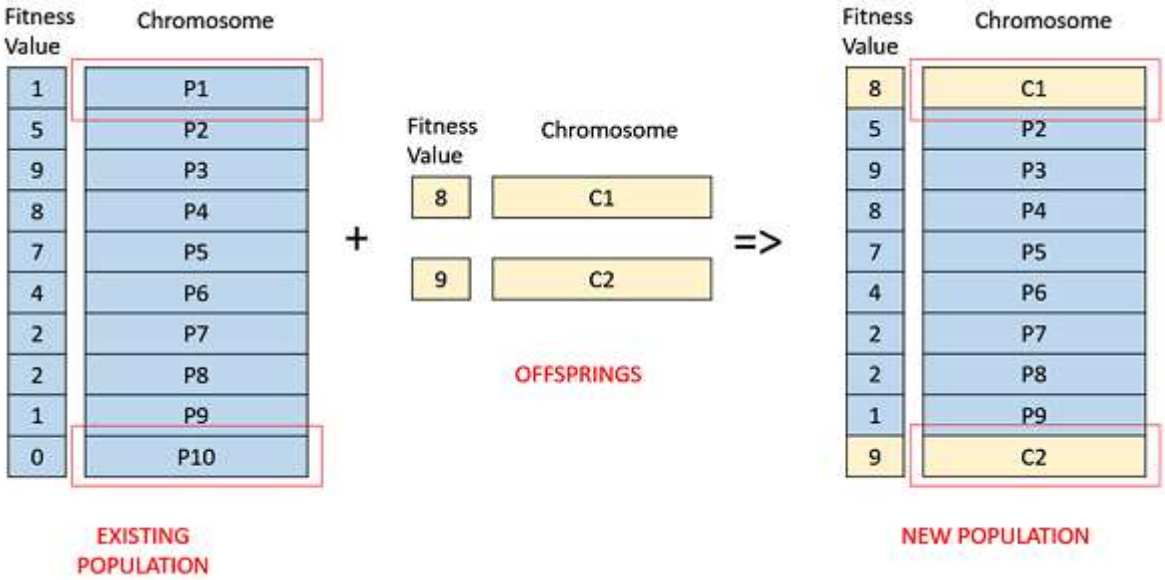
For instance, in the following example, the age is the number of generations for which the individual has been in the population. The oldest members of the population i.e. P4 and P7 are kicked out of the population and the ages of the rest of the members are incremented by one.



Fitness Based Selection

In this fitness based selection, the children tend to replace the least fit individuals in the population. The selection of the least fit individuals may be done using a variation of any of the selection policies described before – tournament selection, fitness proportionate selection, etc.

For example, in the following image, the children replace the least fit individuals P1 and P10 of the population. It is to be noted that since P1 and P9 have the same fitness value, the decision to remove which individual from the population is arbitrary.



Genetic Algorithms - Termination Condition

The termination condition of a Genetic Algorithm is important in determining when a GA run will end. It has been observed that initially, the GA progresses very fast with better solutions coming in every few iterations, but this tends to saturate in the later stages where the improvements are very small. We usually want a termination condition such that our solution is close to the optimal, at the end of the run.

Usually, we keep one of the following termination conditions –

- When there has been no improvement in the population for X iterations.
- When we reach an absolute number of generations.
- When the objective function value has reached a certain pre-defined value.

For example, in a genetic algorithm we keep a counter which keeps track of the generations for which there has been no improvement in the population. Initially, we set this counter to zero. Each time we don't generate off-springs which are better than the individuals in the population, we increment the counter.

However, if the fitness any of the off-springs is better, then we reset the counter to zero. The algorithm terminates when the counter reaches a predetermined value.

Like other parameters of a GA, the termination condition is also highly problem specific and the GA designer should try out various options to see what suits his particular problem the best.

Models Of Lifetime Adaptation

Till now in this tutorial, whatever we have discussed corresponds to the Darwinian model of evolution – natural selection and genetic variation through recombination and mutation. In nature, only the information contained in the individual's genotype can be transmitted to the next generation. This is the approach which we have been following in the tutorial so far.

However, other models of lifetime adaptation – **Lamarckian Model** and **Baldwinian Model** also do exist. It is to be noted that whichever model is the best, is open for debate and the results obtained by researchers show that the choice of lifetime adaptation is highly problem specific.

Often, we hybridize a GA with local search – like in Memetic Algorithms. In such cases, one might choose do go with either Lamarckian or Baldwinian Model to decide what to do with individuals generated after the local search.

Lamarckian Model

The Lamarckian Model essentially says that the traits which an individual acquires in his/her lifetime can be passed on to its offspring. It is named after French biologist Jean-Baptiste Lamarck.

Even though, natural biology has completely disregarded Lamarckism as we all know that only the information in the genotype can be transmitted. However, from a computation view point, it has been shown that adopting the Lamarckian model gives good results for some of the problems.

In the Lamarckian model, a local search operator examines the neighborhood (acquiring new traits), and if a better chromosome is found, it becomes the offspring.

Baldwinian Model

The Baldwinian model is an intermediate idea named after James Mark Baldwin (1896). In the Baldwin model, the chromosomes can encode a tendency of learning beneficial behaviors. This means, that unlike the Lamarckian model, we don't transmit the acquired traits to the next generation, and neither do we completely ignore the acquired traits like in the Darwinian Model.

The Baldwin Model is in the middle of these two extremes, wherein the tendency of an individual to acquire certain traits is encoded rather than the traits themselves.

In this Baldwinian Model, a local search operator examines the neighborhood (acquiring new traits), and if a better chromosome is found, it only assigns the improved fitness to the chromosome and does not modify the chromosome itself. The change in fitness signifies the chromosomes capability to "acquire the trait", even though it is not passed directly to the future generations.

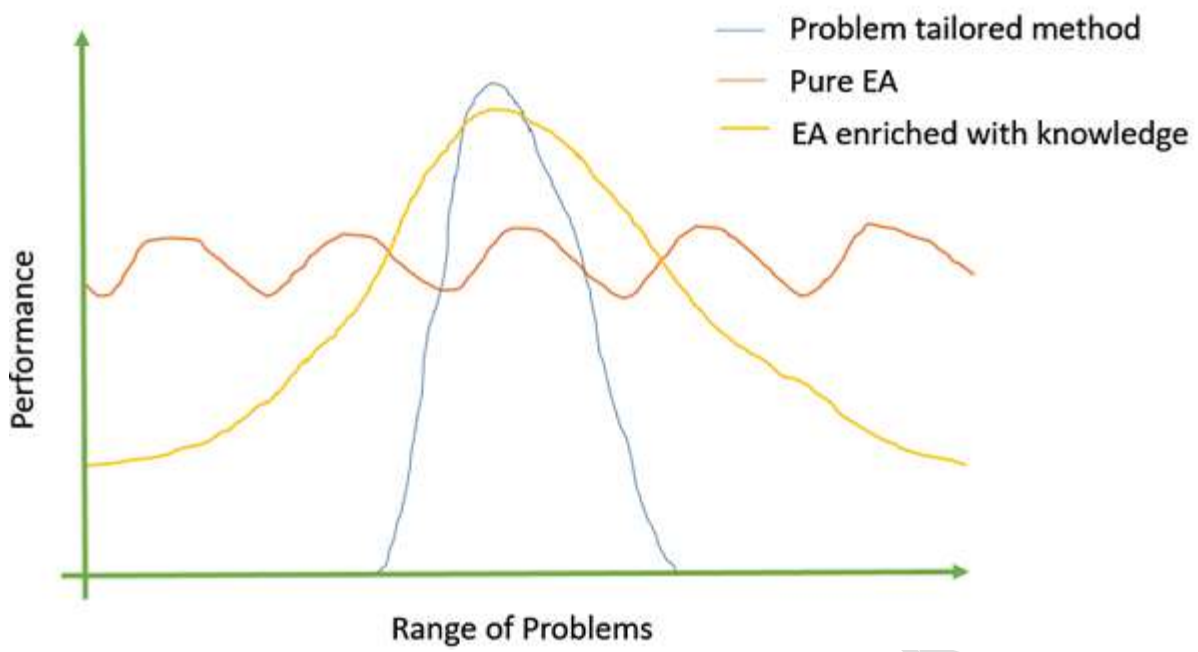
Effective Implementation

GAs are very general in nature, and just applying them to any optimization problem wouldn't give good results. In this section, we describe a few points which would help and assist a GA designer or GA implementer in their work.

Introduce problem-specific domain knowledge

It has been observed that the more problem-specific domain knowledge we incorporate into the GA; the better objective values we get. Adding problem specific information can be done by either using problem specific crossover or mutation operators, custom representations, etc.

The following image shows Michalewicz's (1990) view of the EA –



Reduce Crowding

Crowding happens when a highly fit chromosome gets to reproduce a lot, and in a few generations, the entire population is filled with similar solutions having similar fitness. This reduces diversity which is a very crucial element to ensure the success of a GA. There are numerous ways to limit crowding. Some of them are –

- **Mutation** to introduce diversity.
- Switching to **rank selection** and **tournament selection** which have more selection pressure than fitness proportionate selection for individuals with similar fitness.
- **Fitness Sharing** – In this an individual's fitness is reduced if the population already contains similar individuals.

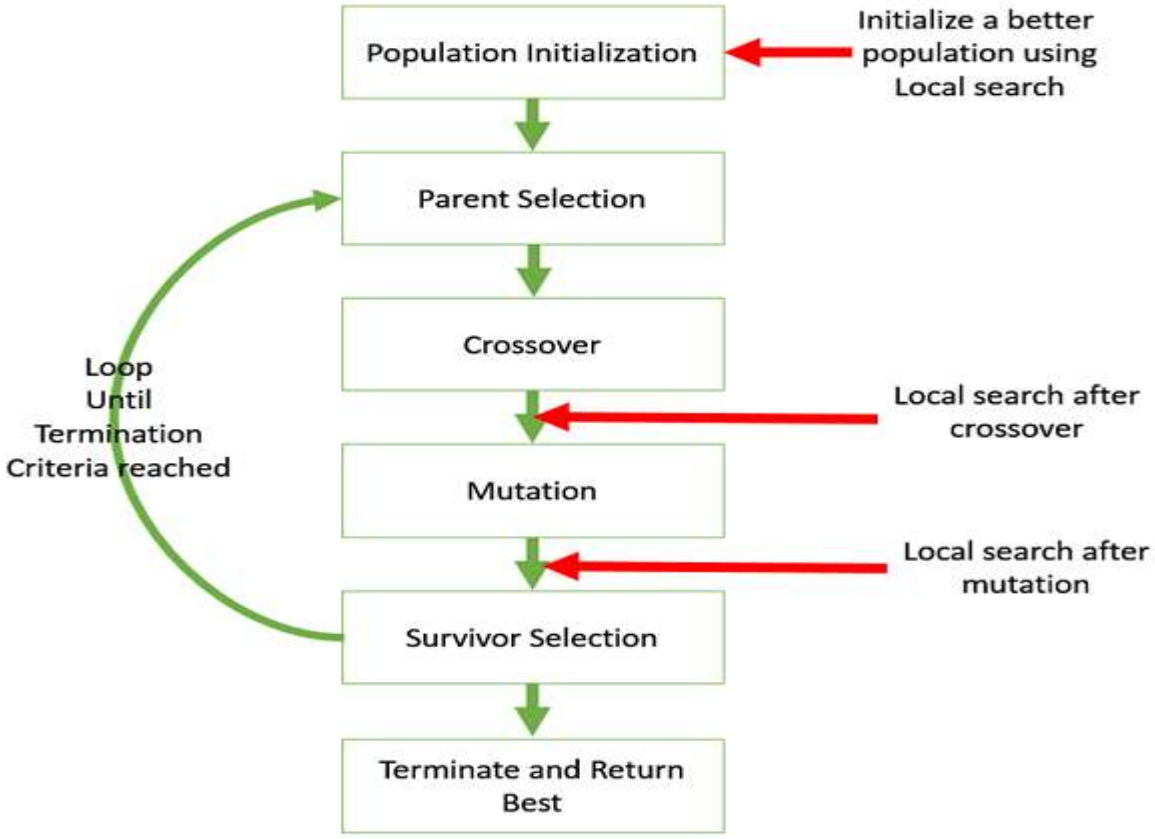
Randomization Helps!

It has been experimentally observed that the best solutions are driven by randomized chromosomes as they impart diversity to the population. The GA implementer should be careful to keep sufficient amount of randomization and diversity in the population for the best results.

Hybridize GA with Local Search

Local search refers to checking the solutions in the neighborhood of a given solution to look for better objective values.

It may be sometimes useful to hybridize the GA with local search. The following image shows the various places in which local search can be introduced in a GA.



Variation of parameters and techniques

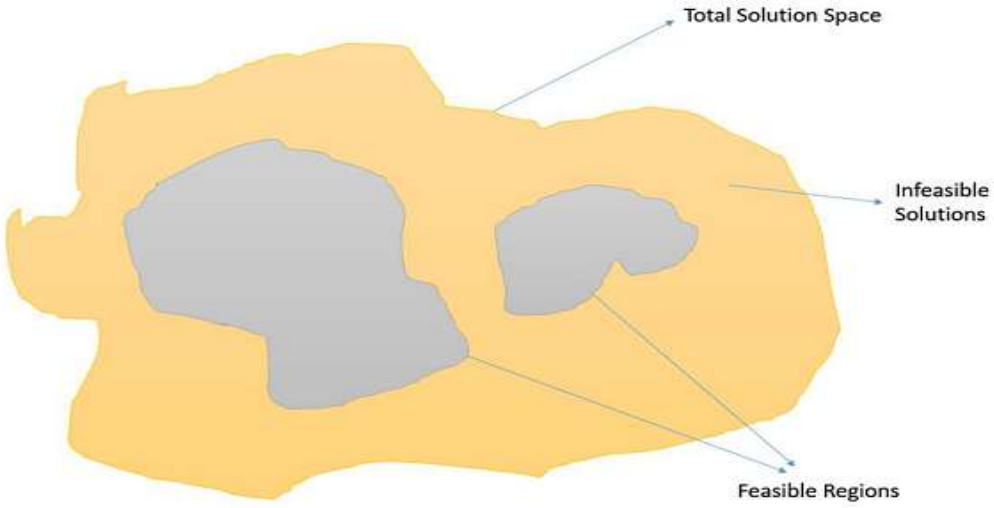
In genetic algorithms, there is no "one size fits all" or a magic formula which works for all problems. Even after the initial GA is ready, it takes a lot of time and effort to play around with the parameters like population size, mutation and crossover probability etc. to find the ones which suit the particular problem.

Genetic Algorithms - Advanced Topics

In this section, we introduce some advanced topics in Genetic Algorithms. A reader looking for just an introduction to GAs may choose to skip this section.

Constrained Optimization Problems

Constrained Optimization Problems are those optimization problems in which we have to maximize or minimize a given objective function value that is subject to certain constraints. Therefore, not all results in the solution space are feasible, and the solution space contains feasible regions as shown in the following image.



In such a scenario, crossover and mutation operators might give us solutions which are infeasible. Therefore, additional mechanisms have to be employed in the GA when dealing with constrained Optimization Problems.

Some of the most common methods are –

- Using **penalty functions** which reduces the fitness of infeasible solutions, preferably so that the fitness is reduced in proportion with the number of constraints violated or the distance from the feasible region.

- Using **repair functions** which take an infeasible solution and modify it so that the violated constraints get satisfied.
- **Not allowing infeasible solutions** to enter into the population at all.
- Use a **special representation or decoder functions** that ensures feasibility of the solutions.

Basic Theoretical Background

In this section, we will discuss about the Schema and NFL theorem along with the building block hypothesis.

Schema Theorem

Researchers have been trying to figure out the mathematics behind the working of genetic algorithms, and Holland's Schema Theorem is a step in that direction. Over the year's various improvements and suggestions have been done to the Schema Theorem to make it more general.

In this section, we don't delve into the mathematics of the Schema Theorem, rather we try to develop a basic understanding of what the Schema Theorem is. The basic terminology to know are as follows –

- A **Schema** is a "template". Formally, it is a string over the alphabet = {0,1,*},

where * is don't care and can take any value.

Therefore, *10*1 could mean 01001, 01011, 11001, or 11011

Geometrically, a schema is a hyper-plane in the solution search space.

- **Order** of a schema is the number of specified fixed positions in a gene.

Schema	Order
***	0
101	3
*11	2
1**	1

- **Defining length** is the distance between the two furthest fixed symbols in the gene.

Schema	Defining Length
****	0
11	1
1*0*	2
1111	3

The schema theorem states that this schema with above average fitness, short defining length and lower order is more likely to survive crossover and mutation.

Building Block Hypothesis

Building Blocks are low order, low defining length schemata with the above given average fitness. The building block hypothesis says that such building blocks serve as a foundation for the GAs success and adaptation in GAs as it progresses by successively identifying and recombining such "building blocks".

No Free Lunch (NFL) Theorem

Wolpert and Macready in 1997 published a paper titled "No Free Lunch Theorems for Optimization." It essentially states that if we average over the space of all possible problems, then all non-revisiting black box algorithms will exhibit the same performance.

It means that the more we understand a problem, our GA becomes more problem specific and gives better performance, but it makes up for that by performing poorly for other problems.

GA Based Machine Learning

Genetic Algorithms also find application in Machine Learning. **Classifier systems** are a form of **genetics-based machine learning** (GBML) system that are frequently used in the field of machine learning. GBML methods are a niche approach to machine learning.

There are two categories of GBML systems –

- **The Pittsburg Approach** – In this approach, one chromosome encoded one solution, and so fitness is assigned to solutions.
- **The Michigan Approach** – one solution is typically represented by many chromosomes and so fitness is assigned to partial solutions.

It should be kept in mind that the standard issue like crossover, mutation, Lamarckian or Darwinian, etc. are also present in the GBML systems.

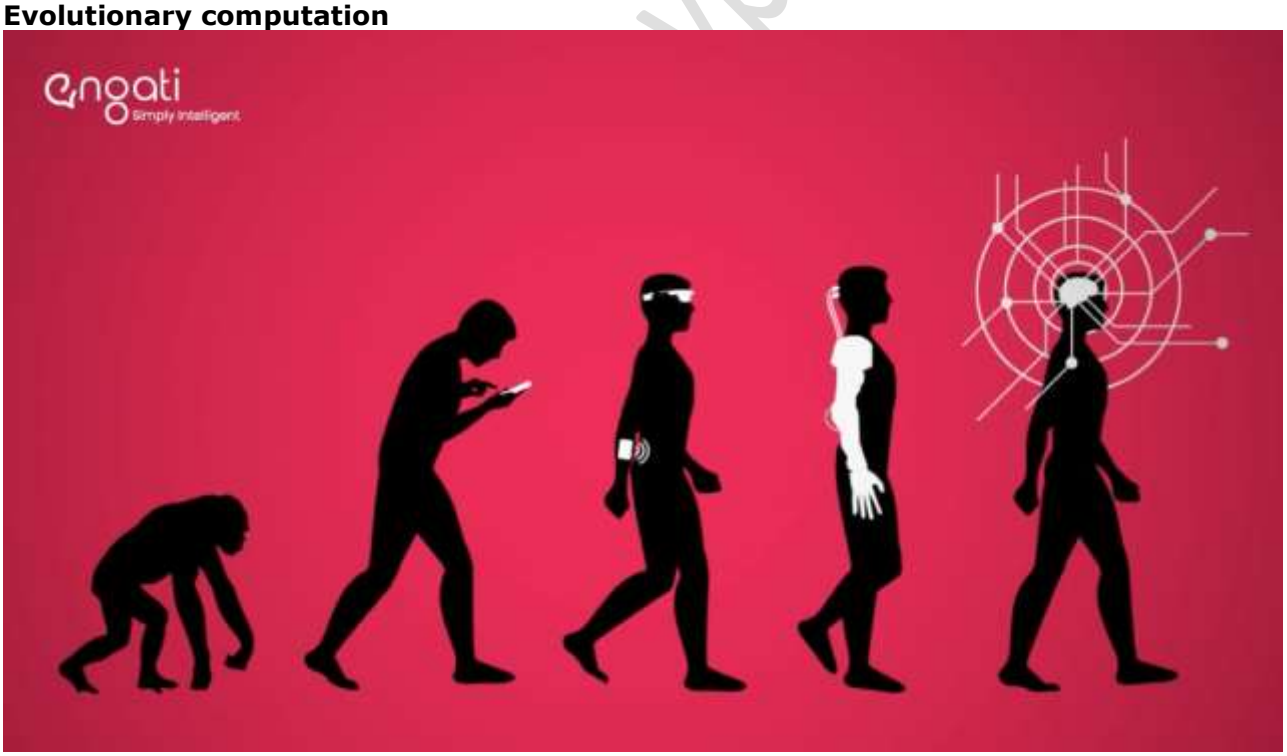
Genetic Algorithms - Application Areas

Genetic Algorithms are primarily used in optimization problems of various kinds, but they are frequently used in other application areas as well.

In this section, we list some of the areas in which Genetic Algorithms are frequently used. These are –

- **Optimization** – Genetic Algorithms are most commonly used in optimization problems wherein we have to maximize or minimize a given objective function value under a given set of constraints. The approach to solve Optimization problems has been highlighted throughout the tutorial.
- **Economics** – GAs are also used to characterize various economic models like the cobweb model, game theory equilibrium resolution, asset pricing, etc.
- **Neural Networks** – GAs are also used to train neural networks, particularly recurrent neural networks.
- **Parallelization** – GAs also have very good parallel capabilities, and prove to be very effective means in solving certain problems, and also provide a good area for research.
- **Image Processing** – GAs are used for various digital image processing (DIP) tasks as well like dense pixel matching.
- **Vehicle routing problems** – With multiple soft time windows, multiple depots and a heterogeneous fleet.
- **Scheduling applications** – GAs are used to solve various scheduling problems as well, particularly the time tabling problem.
- **Machine Learning** – as already discussed, genetics based machine learning (GBML) is a niche area in machine learning.
- **Robot Trajectory Generation** – GAs have been used to plan the path which a robot arm takes by moving from one point to another.
- **Parametric Design of Aircraft** – GAs have been used to design aircrafts by varying the parameters and evolving better solutions.
- **DNA Analysis** – GAs have been used to determine the structure of DNA using spectrometric data about the sample.
- **Multimodal Optimization** – GAs are obviously very good approaches for multimodal optimization in which we have to find multiple optimum solutions.
- **Traveling salesman problem and its applications** – GAs have been used to solve the TSP, which is a well-known combinatorial problem using novel crossover and packing strategies.

Evolutionary computation



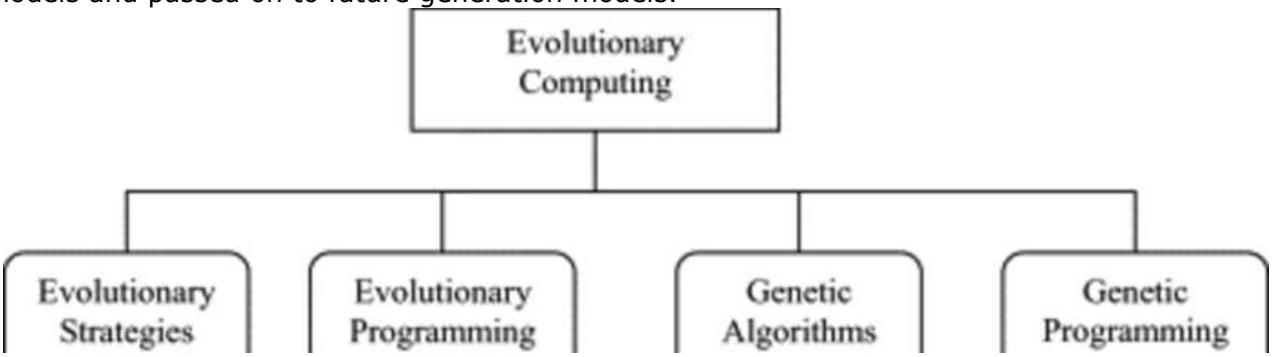
What is evolutionary computation?

Evolutionary computation is a branch of Artificial Intelligence and is used heavily for complex optimization problems and also for continuous optimization.

Evolutionary computation techniques are used to handle problems that have far more variables than what traditional algorithms can handle.

These computational models employ evolutionary algorithms that essentially use evolutionary processes for the purpose of solving such complex problems. They use evolutionary principles like inheritance from

effective previous generation models, and natural selection, where the traits from the most effective models and passed on to future generation models.

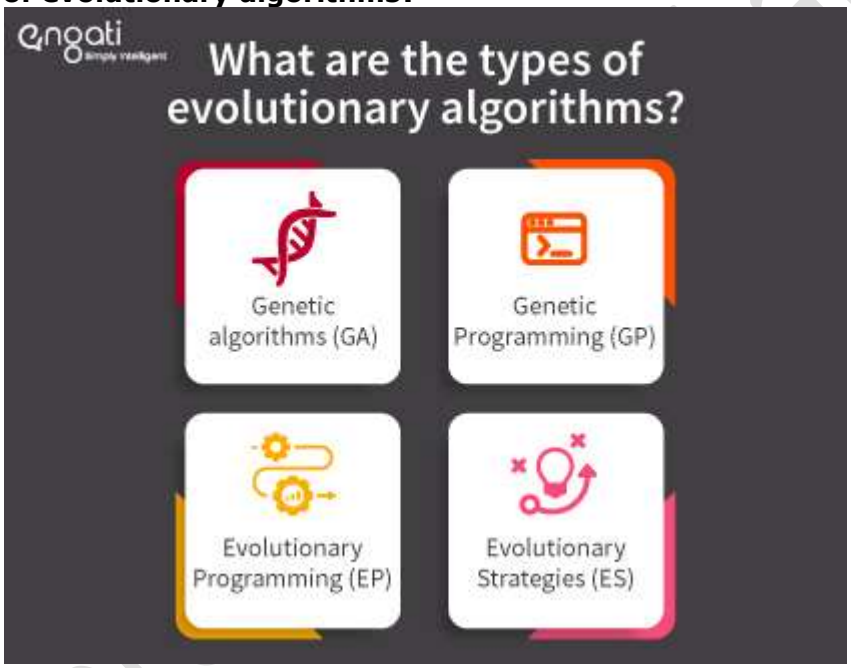


Why do we use evolutionary computation?

Since evolutionary computing has the ability to produce tightly optimized solutions for a wide range of problems, they are extensively used in computer science. There are even variants that are created and used specifically for particular data structures and families of problems.

This branch of artificial intelligence is also employed in evolutionary biology for studying common aspects of general evolutionary processes.

What are the types of evolutionary algorithms?



There are various types of evolutionary algorithms. Here are the most significant ones:

- Genetic algorithms (GA)
- Genetic Programming (GP)
- Evolutionary Programming (EP)
- Evolutionary Strategies (ES)

Many more evolutionary algorithms also exist. These include Gene Expression Programming, Differential Evolution, Learning Classifier Systems, and Neuroevolution.

Genetic algorithms (GA)

Genetic algorithms are the most popular type of evolutionary algorithms. They find solutions to problems as strings of numbers. Most of these strings are binary, but the most effective ones tend to show something about the problem in question.

These algorithms make use of operators like mutation and recombination. Sometimes they use both operators together.

One of the uses of genetic algorithms is selecting the right combination of variables to build a predictive model. Selecting the right subset of variables is essentially a combinatory and optimization problem.

The advantage of genetic algorithms is that it makes it possible for the best solution to emerge from the best of prior solutions. It improves the selection over time.

The whole idea behind genetic algorithms is to combine the differnet solutions generation after generation so that it can extract the best genes or variables from each solution. It helps creating better fitted individuals.

Genetic algorithms are also used for hyper-tuning parameters, finding the maximum or minimum of a function, or the search for a correct neural network architecture (Neuroevolution). It is also used in feature selection.

The idea of genetic algorithms (GA) is to generate a few random possible solutions that represent different variables, and then combine the best possible solutions in an iterative process. The basic genetic algorithm operations are selection (picking the most fitted solutions in a generation), cross-over (creating two new individuals, based on the genes of solutions), and mutation (changing a gene randomly in an individual).

Genetic Programming (GP)

Here, the solutions to problems are computer programs. The ability of these computer programs to solve computational problems is what determines their fitness.

Genetic Programming (GP) is essentially an automatic programming technique which favors the evolution of computer programs that solve or at least approximately solve problems. It involves essentially 'breeding' programs by continuously improving an initially random set of programs.

Improvements are made by stochastic variation of programs and selection in line with some predefined criteria for judging the quality of a solution. Programs of genetic programming systems essentially evolve to solve predescribed automatic programming and machine learning problems.

In its essence, genetic programming is a heuristic search technique that is commonly called 'hill climbing'. It involves searching for an optimal or at least a suitable program among the space of all programs.

Evolutionary Programming (EP)

This is not too different from Genetic Programming. However, in Evolutionary Programming, the programs that need to be optimized have a fixed structure, while the numeric parameters can evolve.

This evolutionary algorithm paradigm was first used by Lawrence J. Fogel in 1960 in an attempt to use simulated evolution as a learning process seeking to create artificial intelligence. He used finite-state machines as predictors and evolved them. Right now, evolutionary programming is a wide evolutionary computing dialect that has no fixed structure or representation. It is becoming increasingly difficult to differentiate evolutionary programming from evolutionary strategies.

The main operator of evolutionary programming is mutation. In evolutionary programming, members of the population are seen as part of a specific species rather than members of the same species. Every parent generates an offspring by using a (μ + μ) survivor selection.

Evolutionary Strategies (ES)

Evolutionary strategies usually work by making use of self-adaptive mutation rates. They work with vectors of real numbers as representations of solutions.

Evolutionary strategies are optimization techniques that are based on the ideas of evolution. They use natural problem-dependent representations and mainly make use of mutation and selection as search operators. The operators are applied in a loop, an iteration of which is known as a generation. The sequence of generations continues till a termination criterion is met. Most evolutionary algorithms work on a genotype level, but evolutionary strategies work on a behavioral level.

Since the physical expression is coded directly, an individual's genes are not mapped to its physical expression.

This approach is followed to give rise to a strong causality so that a small change in the coding gives rise to a small change in the individual and a large change in the coding causes a large change in the individual.

How does evolutionary computing work?

At the initial stage of the evolutionary computation process, an initial batch of possible solutions is created.

After that, the system tests the solutions proposed and stochastically removes the solutions that do not perform well, thus refining the model.

It introduces tiny randomized changes to future generations of the model, and over generations the solutions are refined to a high degree.

The solutions at the end of the evolutionary computing process are highly optimized.

Introduction to Swarm Intelligence

Swarm Intelligence (S.I.) was introduced by **Gerardo Beni and Jing Wang in the year 1989**. S.I. simply means using the knowledge of collective objects (people, insects, etc.) together and then reaching the optimized solution for a given problem. ***"Swarm" means a group of objects (people, insects, etc.)***. In other words, let's say we give a problem statement to a single person and tell him or her to go through this problem and then give the solution, then this means that we will consider the solution of that particular person only, but the problem is that the solution given by that person may not be the best solution or maybe, that solution is not good for others. So to avoid that, what we do is we give that problem to a certain amount of people together (swarm) and ask them to reach the best solution possible for that problem, and then computing all the responses together to reach the best solution possible, so here we are using the knowledge of the group as a whole to reach to the best solution or optimized

solution for that problem and that solution will be good for all of them individually too, so that is the idea behind swarm intelligence.

***Note**: In swarm intelligence, each individual (object) in the group is independent of others, each individual is responsible for their own contribution to solve that problem regardless of what others are doing.*

Let's take an example ***to prove that the collective knowledge of objects is better than any individual object***.

Example

Let's say we have a jar containing 500 marbles in that. The question is without touching the jar a person needs to predict how many marbles are in that jar. Suppose we take only one response from a person and it predicts that according to him the jar contains 400 marbles. So by this result, we can conclude that this estimation of that person is not very bad since the ***difference (error) is of 100*** only, but this might not be the best solution, we can optimize this even more. So now what we will do is instead of taking response from only one person we will be taking response from 10 people let's say. Let 'P' denote a person therefore the responses are as follows:

1	2	3	4	5	6	7	8	9	1
0	5	5	0	8	9	2	9	1	5

So after collecting the responses from 10 different individuals we can take the average of their responses.

Average:

(400 + 450 + 550 + 600 + 480 + 390 + 520 + 490 + 510 + 450) / 10

Average = 4840 / 10

= 484 (marbles in the jar)

Now from this, we can say that from the collective predictions from 10 different persons we have reached a more optimal answer that is 484 marbles in the jar. We are very close to the actual result of 500 marbles in the jar, here in this case the ***difference (error) reduces to only 16 marbles as compared to the previous error which was 100***. So that is the main idea behind swarm intelligence, that is to use the collective knowledge of objects.

Why Swarm Intelligence?

The answer to this is simple now and we proved this in our previous marble example, which is collecting the answers (responses) from different objects individually and then computing all responses as a whole to a solution that best fits our given problem. So here, with this approach, we are having a more optimized solution for a given problem and that is the reason why swarm intelligence came into the picture, because of this reason we can use it in different scenarios of life e.g. Forecasting, Which policy is good for the business, etc. So simply we are using the **'Brain of Brains'** to reach the solution for a given problem. If we will observe in our surroundings (nature) then we will be able to find many examples of swarm intelligence like ***'ant colony', 'swarm of bees', 'flock of birds'*** etc. and in reality, also the idea of swarm intelligence was taken from nature only. Some are explained below:

Ant Colony

If we will observe closely then the ants also follow the principle of swarm intelligence, for example, to build the home they collect mud particles from the surroundings and individually have a responsibility to build their home. They communicate through signals and pheromones (ants use this for tracing other ants) and regardless of what other ants are doing, an individual ant is responsible for only its own contribution to build the home. Similarly when they search for food then at first they search individually for food leaving the pheromones behind and once they find the food source then that ant communicates with other ants and then other ants can trace it and follow that path to reach the source of food instead of just randomly searching food in different locations every time.

Swarm of Bees

Bees also use the same principle for their survival that is when they search for the place like where they can build their hive, then the task of each bee is to consider several parameters that the hive which will be built should be on good height to avoid predators, should be near water resource, should be near pollens (flowers to collect nectar), etc. then they use their collective research and finally a place is decided that where the hive would be built considering all those parameters and they reach the best solution for that problem.

Artificial Swarm Intelligence (ASI)

It is also known as Human Swarm. Here also the idea is same that we randomly make some of the persons participating in a real-time system and tell them to find the solution for that particular problem

individually, the final solution is then computed after taking responses from each participant and the final solution is presented which is more optimized as compared to the solution taking from only one participant.

Applications of Swarm Intelligence

- Used in military services.
- NASA is generating the idea to use swarm intelligence for planetary mapping.
- Used in Data Mining.
- M. Anthony Lewis and George A. Bekey presented the idea that with the help of swarm intelligence we can control nanobots in our body to kill cancer tumors.
- Used in business to reach better financial decisions etc.

Introduction

Advances in scalable computing and artificial intelligence have developed swarm intelligence approaches. In fact, swarm intelligence algorithms use the cooperative and group behavior of social organisms in nature.

In this tutorial, we'll look at what this swarm intelligence means and what it does. Then, we'll go through different real-world examples and applications of swarm intelligence algorithms.

Finally, we'll choose an example to demonstrate the importance of the swarm intelligence approach using one of the most popular swarm intelligence-based optimization algorithms.

2. What's Swarm Intelligence

Swarm intelligence is an artificial or natural intelligence technique. It is based on studying collective behavior in decentralized and self-organized systems. Gerardo Benny and Joon Wang introduced swarm intelligence in 1989 in the context of cellular robotics systems.

2.1. Principles of Swarm Intelligence

Let's discuss the swarm intelligence principles:

- **Awareness:** Each member must be aware of their surroundings and abilities
- **Autonomy:** To self-coordinate, each member must operate as an autonomous master (not as a slave)
- **Solidarity:** When a task is completed, members should autonomously look for a new task
- **Expandability:** The system must permit dynamic expansion where members are seamlessly aggregated
- **Resiliency:** When members are removed, the system must be self-healing

2.2. Real-World Examples

We have various intelligence examples in nature such as Ant colonies, Bee beehives, Fish schooling, Bird flocking, Bacterial growth, and microbial intelligence.

Also, there are biological advantages of swarm intelligence. For example, birds steal information using up to a fifth less energy than those that fly solo. In addition, Swarm intelligence is modeled for the purpose of understanding microscopic (global) transformations. Furthermore, it allows getting ideas for artificial systems the similar proprieties.

Besides, there are two main development areas of swarm intelligence:

- **Particle Swarm Optimization:** One of the most well-known swarm intelligence-based optimization techniques. Particle swarm optimization was modeled by the social behavior of animals and insects. In this context, every individual swarm member is handled as a particle. Cooperation and learning enable the collective intelligence of these dispersed particles.
- **Ant Colony Optimization:** Based on the social instincts of actual ants for their community, the ant colony optimization approach is an essential component of swarm intelligence. This algorithm enables them to cooperate to accomplish a common objective. Since they are all gathered in one place, the ants must locate the food and carry it back to the colony.

2.3. Applications

In addition to its use in traditional optimization issues, swarm intelligence has applications in the acquisition of library items, communications, the categorization of medical datasets, dynamic control, the planning of heating systems, and the tracking and prediction of moving objects. Furthermore, swarm intelligence may be used in many different areas of basic research, engineering, business, and the social sciences.

3. Swarm Intelligence Algorithm Example

In this part, we choose to describe Particle Swarm Optimization.

Particle swarm optimization has been regarded as one of the most effective optimization approaches for decades due to its ease of use, the number of parameters that may be utilized for modification, speedy convergence, and scalability. **In addition, particle swarm optimization is the first swarm-based algorithm. This algorithm allows the people in a basic living structure to join forces to create a more intelligent structure.**

3.1. Particle Swarm Optimization Algorithm

Let's see how conventional particle swarm optimization works:

Algorithm 1: Particle swarm optimization algorithm

```
Initialisation: v_i, x_i, p_best, g_best, iteration
Generate random particles P
for each particle (i) do
    Determine fitness function (f_i)
    (p_best), (g_best)
end
while iteration do
    for each particle (i) do
        Update(v_i, x_i)
        if x_i > limit then
            x_i = limit
        end
        Calculate fitness function (f_i)
        Update (p_best), (g_best)
    end
end
```

According to the particle swarm optimization algorithm stages, there are three crucial elements that stand out: the , , particle, and fitness function parameters. The correct specification of the fitness function is a prerequisite for using meta-heuristic algorithms. Also, the iteration allows us to repeat the algorithm steps until we get the best solution. These algorithms work by transforming issues into fitness functions that display how far away the particles are from the ideal solution:

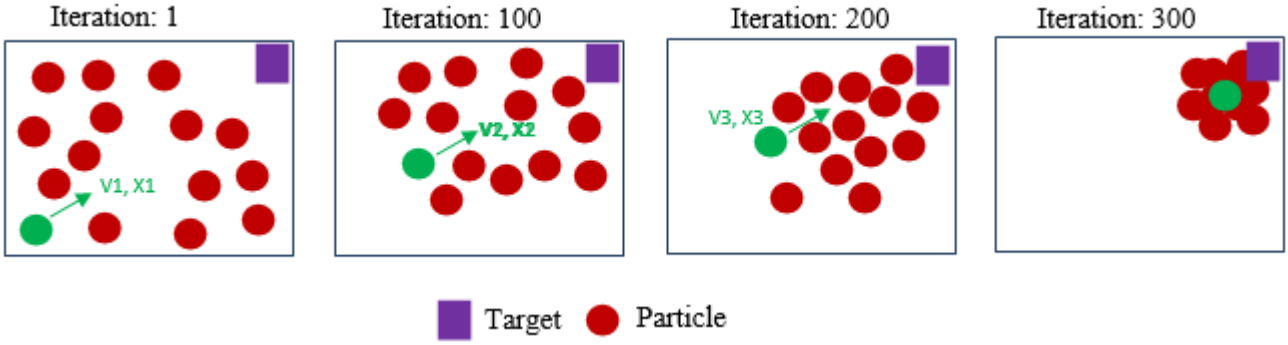
is the current velocity of the particle (Equation. 1) and is the position (Equation. 2). Within the equations. and ; is the size of the problem, and is the number of particles. Also, and present the author's and the world's best weights; and and are random values in the range .

Besides, the optimal solution, or , is the separation between the position of each particle and the food source. The closest distance the swarm has come to the food source in one iteration is .

Generally, these two elements influence the convergence of the particle. These two factors are crucial because they determine whether the particle will eventually reach a solution. Moreover, the number of particles is a parameter that directly affects the solution. **Indeed, since the number increases, the result improves but the algorithm runs more slowly.**

3.2. Particle Swarm Optimization

In particle swarm optimization, the particles move to a new position with a new velocity with each iteration, as shown below with the different number of iterations; 1, 100, 200, and 300. Therefore, each time they come closer to the goal, that is, to the solution to the problem:



The fact that optimal values can be obtained with only two parameters in the algorithm is an indication of the strength of this algorithm. For this reason, it is preferred in many different engineering problems due to its ease of application, simple structure, and efficient production results.

Introduction to Ant Colony Optimization

What is Algorithm?

Algorithms are processes or optimized solutions for any complex problems. There is always a principle behind any algorithm design. Sometimes, these algorithms are designed from natural laws and events, and evolutionary algorithms are the example of these algorithms.

This algorithm uses natural events and behavior as it is to get the low-cost and best possible solution to a complex problem.

There are a lot of algorithms based on natural behavior, and they are called metaheuristics. Metaheuristics are made of two words: meta, which means one level above, and heuristics, which means to find.

Particle Swarm Optimization and **Ant Colony Optimization** are examples of these swarm intelligence algorithms. The objective of the swarm intelligence algorithms is to get the optimal solution from the behavior of insects, ants, bees, etc.

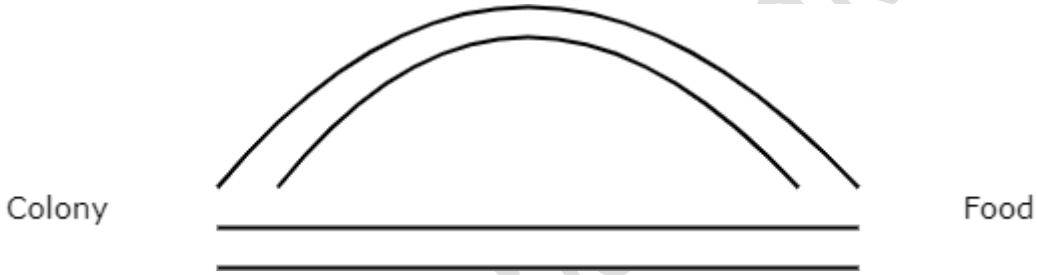
Principle of Ant Colony Optimization

This technique is derived from the behavior of ant colonies. Ants are social insects that live in groups or colonies instead of living individually. For communication, they use pheromones. Pheromones are the chemicals secreted by the ants on the soil, and ants from the same colony can smell them and follow the instructions.

To get the food, ants use the shortest path available from the food source to the colony. Now ants going for the food secret the pheromone and other ants follow this pheromone to follow the shortest route. Since more ants use the shortest route so the concentration of the pheromone increase and the rate of evaporation of pheromone to other paths will be decreased, so these are the two major factors to determine the shortest path from the food source to the colony.

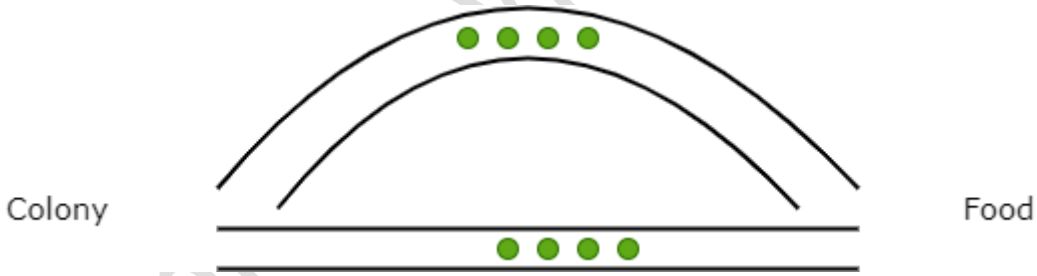
We can understand it by following steps:

Stage 1:



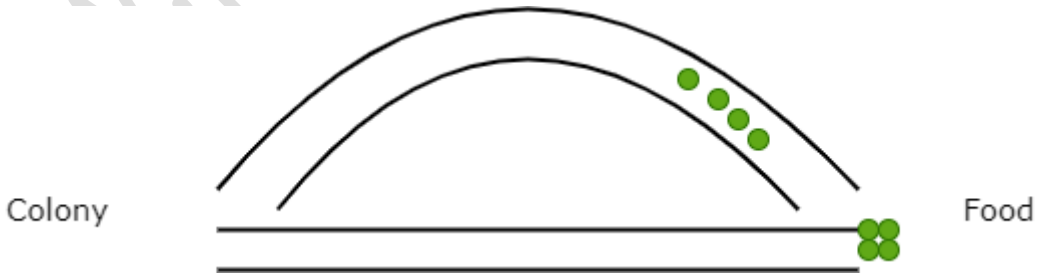
In this stage, there is no pheromone in the path, and there are empty paths from food to the ant colony.

Stage2:



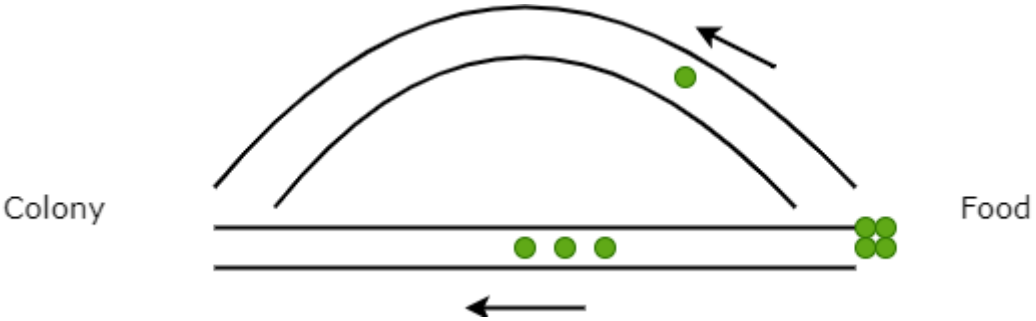
In this stage, ants are divided into two groups following two different paths with a probability of 0.5. So we have four ants on the longer path and four on the shorter path.

Stage 3:



Now, the ants which follow the shorter path will react to the food first, and then the pheromone concentration will be higher on this path as more ants from the colony will follow the shorter path.

Stage 4:



Now more ants will return from the shortest path, and the concentration of pheromones will be higher. Also, the rate of evaporation from the longer path will be higher as fewer ants are using that path. Now more ants from the colony will use the shortest path.

Algorithm Design

Now the above behavior of the ants can be used to design the algorithm to find the shortest path. We can consider the ant colony and food source as the node or vertex of the graph and the path as the edges to these vertices. Now the pheromone concentration can be assumed as the weight associated with each path.

Let's suppose there are only two paths which are P_1 and P_2. C_1 and C_2 are the weight or the pheromone concentration along the path, respectively.

So we can represent it as graph G(V, E) where V represents the Vertex and E represents the Edge of the graph.

Initially, for the i^{th} path, the probability of choosing is:

$$Pi = \frac{Ci}{C1+C2};\ where\ i = 1,2$$

If $C_1 > C_2$, then the probability of choosing path 1 is more than path 2. If $C_1 < C_2$, then Path 2 will be more favorable.

For the return path, the length of the path and the rate of evaporation of the pheromone are the two factors.

1. Concentration of pheromone according to the length of the path:

$$Ci = Ci + \frac{K}{Li}$$

Where L_i is the length of the path and K is the constant depending upon the length of the path. If the path is shorter, concentration will be added more to the existing pheromone concentration.

2. Change in concentration according to the rate of evaporation:

$$Ci = (1 - v) * Ci$$

Here parameter v varies from 0 to 1. If v is higher, then the concentration will be less.

Pseudo Code:

```
1. Procedure ACO:
2.     Initialize the necessary parameters and pheromone concentration;
3.     while not termination do:
4.         Generate initial ant population;
5.         Calculate the fitness values for each ant of the colony;
6.         Find optimal solution using selection methods;
7.         Update pheromone concentration;
8.     end while
9.  end procedure
```

Ant Colony optimization is used in various problems like the Travelling Salesman Problem etc.

Ant Colony Optimization (ACO) Algorithm to solve Numericcal Optimization Problem

1. Introduction

In the 1990's, Ant Colony Optimization was introduced as a novel nature-inspired method for the solution of hard combinatorial optimization problems. The Ant Colony Algorithms family, in swarm intelligence methods, and it constitutes some metaheuristic optimizations [1]. Initially proposed by Marco Dorigo in 1992 in his PhD thesis, the first algorithm was aiming to search for an optimal path in a graph, based on the behavior of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of numerical problems, and as a result, several problems have emerged, drawing on various aspects of the behavior of ants. From a broader perspective, ACO performs a model-based search and shares some similarities with estimation of distribution algorithms. The ant colony optimization algorithm (ACO) is a probabilistic technique for solving computational problems which can be

reduced to finding good paths through graphs [2]. Artificial Ants stand for multi-agent methods inspired by the behavior of real ants. The pheromone-based communication of biological ants is often the predominant paradigm used. Combinations of Artificial Ants and local search algorithms have become a method of choice for numerous optimization tasks involving some sort of graph, e.g., vehicle routing and internet routing. The burgeoning activity in this field has led to conferences dedicated solely to Artificial Ants, and to numerous commercial applications by specialized companies such as AntOptima. Ant colony optimization is a class of optimization algorithms modeled on the actions of an ant colony. Artificial 'ants' (e.g. simulation agents) locate optimal solutions by moving through a parameter space representing all possible solutions. Real ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions.[4] One variation on this approach is the bees algorithm, which is more analogous to the foraging patterns of the honey bee, another social insect [3].

In the natural world, ants of some species (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but instead to follow the trail, returning and reinforcing it if they eventually find food (see Ant communication). Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength [4]. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over more frequently, and thus the pheromone density becomes higher on shorter paths than longer ones. Pheromone evaporation also has the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained. The influence of pheromone evaporation in real ant systems is unclear, but it is very important in artificial systems. The overall result is that when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads to many ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve [5].

2. Life Cycle of Ant Colony Optimization Algorithm

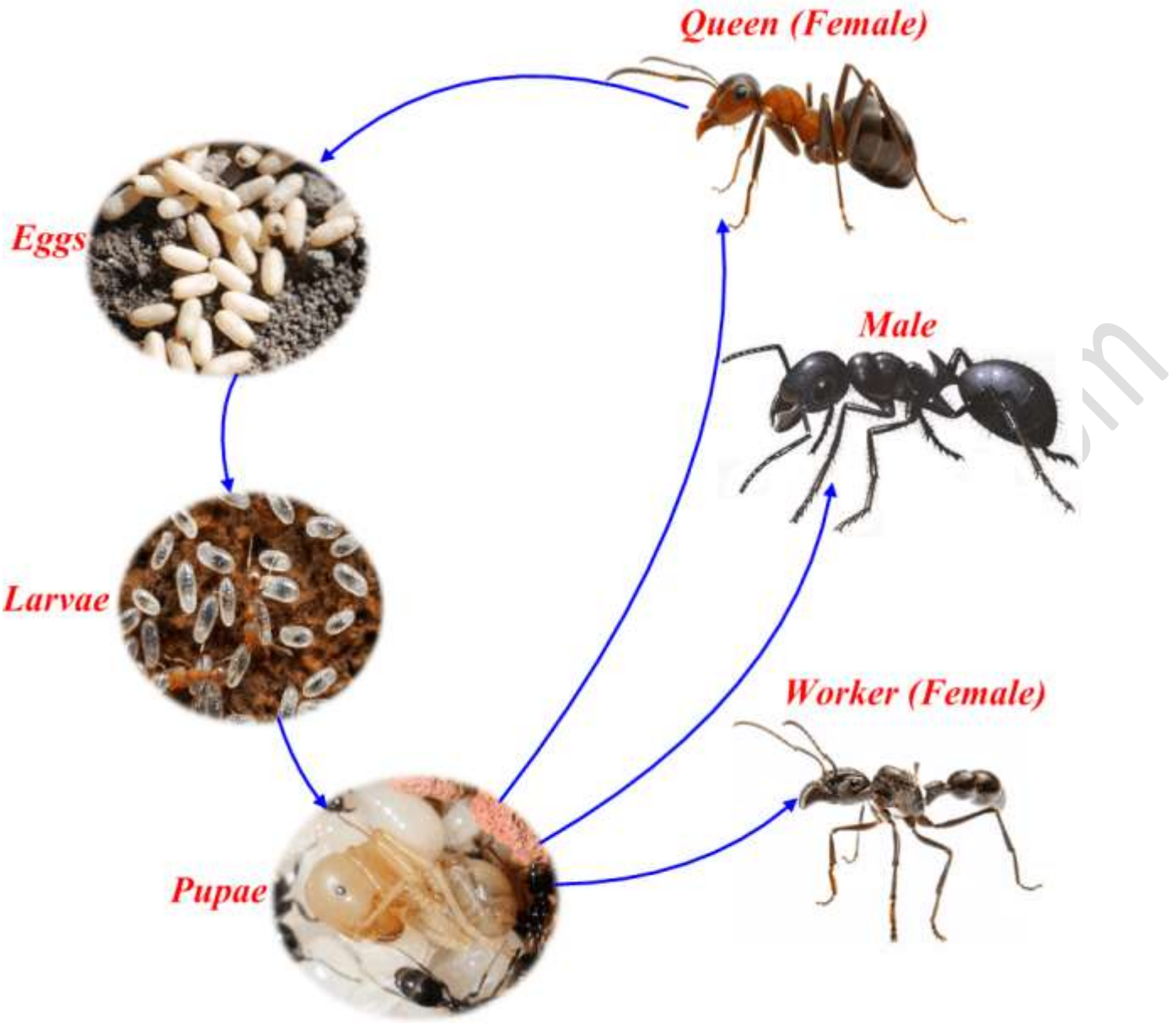


Fig1: Life Cycle of ACO Algorithm

The lifecycle of ants is based on 4 stages of the development: egg → larva → pupa → adult. Eggs are of microscopic size and white colour. Next stage in the development is larva – larvae are legless and fragile and need to be protected/taken care of by worker ants. Pupae are similar to adults, but they are immobile, soft and of a white colour. The last stage of the development is an adult ant. Caste system is remained in colonies and we can distinguish two types of ants: 1. Reproductive – queen(s) and males and 2. Non-reproductive – workers and immature (ants on earlier stages of development) [6].

Colonies are structured and the division of labour is crucial for the colony to survive. The queen is responsible for laying eggs. Workers are the most common ants seen by people and have a wide range of tasks. They gather and storage food, bring it to immature and defend the nest and clean it from the remnants.

The queen is the most important ant and her function is to lay thousands of eggs to ensure the continuity of the colony. Queens mate only once and then are able to reproduce throughout the while life. Therefore males are only needed to mate the queen and most of them die after mating [7].

2.1. Structure of Ant

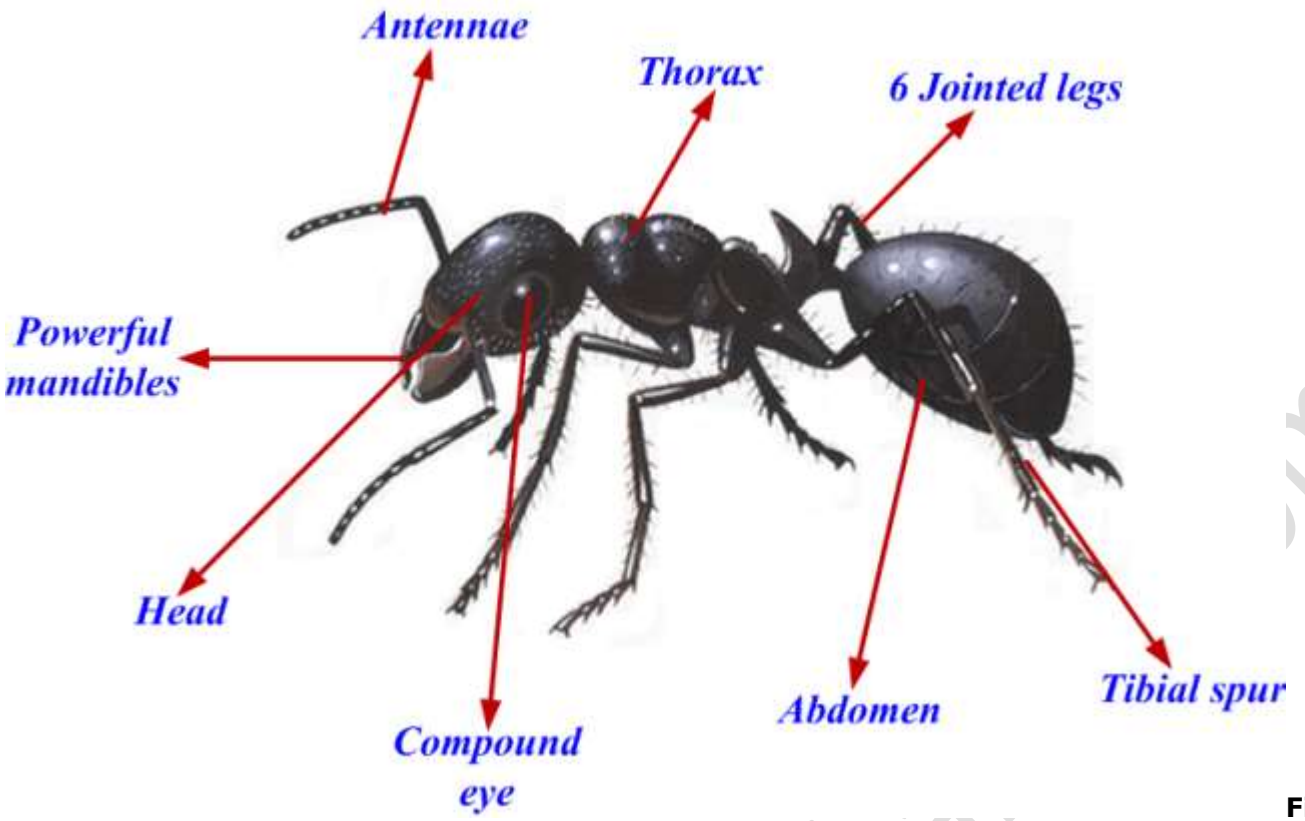


Fig2:

Basic Structure of Ant

Ants, like all insects, have jointed legs, three body parts (the head, thorax and abdomen), a pair of antennae, and a hard exoskeleton. The exoskeleton is made up of a material that is very similar to our fingernails. Ants range in color from yellow to brown to red to black. Some ants have a stinger and some can even inject poisonous acid from the stinger (the stinger is at the tip of the abdomen, the rear body segment) [8]. Ants can also bite using their jaws (mandibles). Ants range in size from about 0.08 inch (2 mm) to up to about 1 inch (25 mm) long.

Abdomen – The abdomen is the segmented tail area of an ant. It contains the heart,Malpighian tubules, reproductive organs, and most of the digestive system (foregut, hindgut and rectum). It is protected by an exoskeleton.

Antennae – Ants have two jointed antennae. They are sensory appendages attached to the head.

Compound eye – Ants have two compound eyes. These eyes are made up of many hexagonal lens/corneas which focus light from each part of the insect's field of view onto a rhabdome (the equivalent of our retina) [9].

Head – The head of an ant (or any insect) is the location of its brain, two compound eyes, its proboscis, pharynx (the start of the digestive system), the point of attachment of its two antennae, etc.

Jointed Leg – Ants, like all insects, have six jointed legs.

Thorax – The thorax is the chest area of an insect (including ants). The thorax is divided into three segments; on each segment is a pair of legs. The thorax contains the muscles that make the legs move [10].

3. Ant Colony Optimization Algorithm (ACO)

The inspiring source of ACO is the foraging behavior of real ants. When searching for food, ants initially explore the area surrounding their nest in a random manner. As soon as an ant finds a food source, it evaluates it and carries some food back to the nest. During the return trip, the ant deposits a pheromone trail on the ground. The pheromone deposited, the amount of which may depend on the quantity and quality of the food, guides other ants to the food source. Indirect communication among ants via pheromone trails enables them to find shortest paths between their nest and food sources. This capability of real ant colonies has inspired the definition of artificial ant colonies that can find approximate solutions to hard combinatorial optimization problems. The central component of ACO algorithms is the pheromone model, which is used to probabilistically sample the search space [11]. ACO is one of the adaptive meta-heuristic optimization methods inspired by nature which include simulated annealing, GA, and tabu search. ACO is distinctly different from other meta-heuristic methods in that it constructs an entire new solution set (colony) in each generation, while others focus on improving the set of solutions or a single solution from previous iterations. ACO was inspired by the behavior of real ants. Ethologists have studied how blind animals, such as ants, could establish shortest paths from their nest to food sources. The medium that is used to communicate information among individual ants regarding paths is pheromone. A moving ant lays some pheromone on the ground, thus marking the path. The pheromone, while gradually dissipating over

time, is reinforced as other ants use the same trail. Therefore, efficient trails increase their pheromone level over time while poor ones reduce to nil [12]. The characteristics of ant colony optimization include:

- A method to construct solutions which balances pheromone trails (characteristics of past solutions) with a problem-specific heuristic (normally, a simple greedy rule),
- Amethod to both reinforce & evaporate pheromone.
- Local (neighborhood) search to improve solutions.

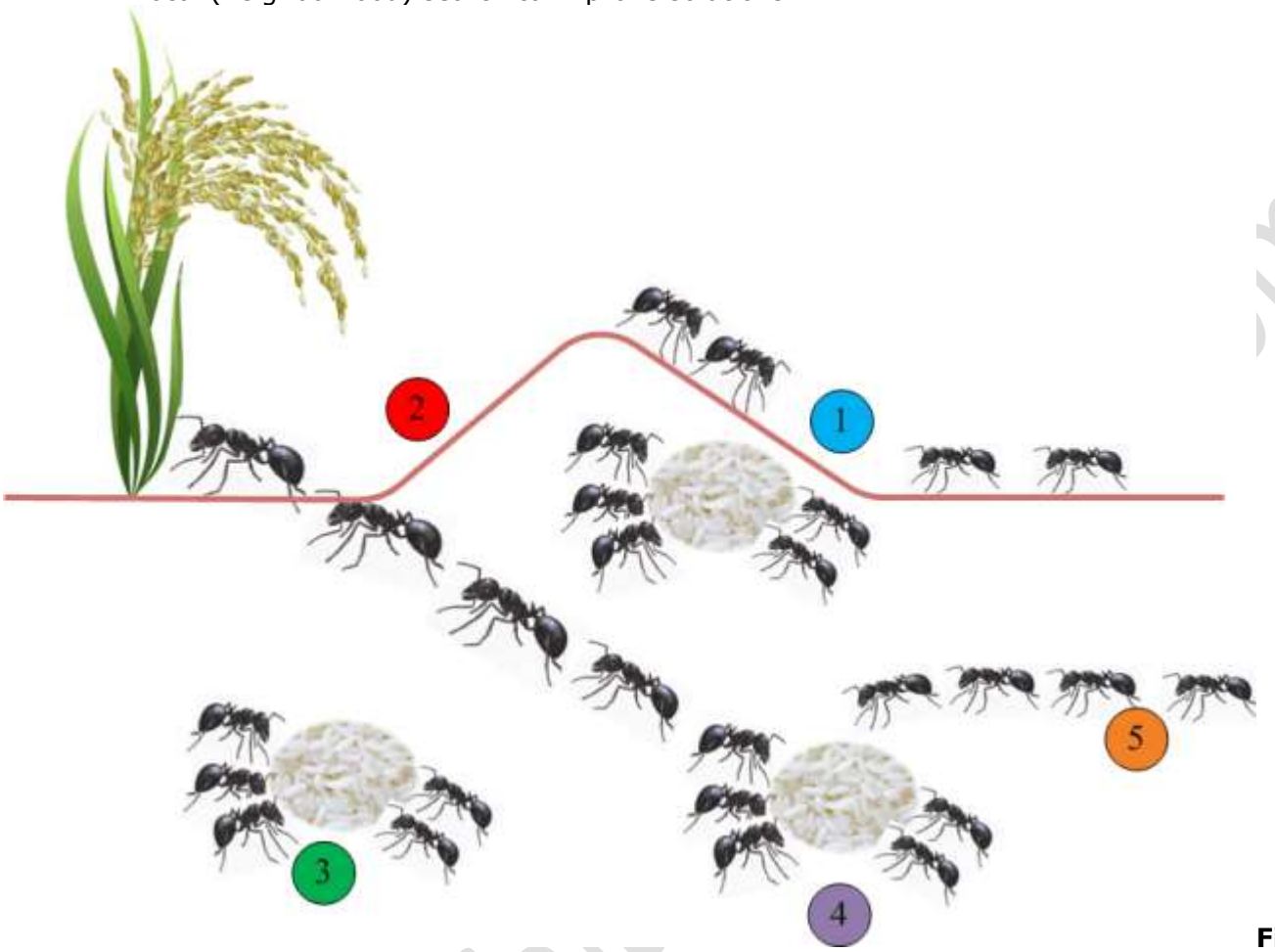


Fig3: ACO

ALgorithm

3.1. Steps for Ant Colony Optimization Algorithm

- Initialization process
- Evaluation of the selected subsets
- Construction process
- Update process
- Decision process

3.1.1. Initialization Process

Initialization of the population is commonly done by seeding the population with random values. The fitness value is proportional to the performance measurement of the function being optimized. The calculation of fitness values is conceptually simple [13]. It can, however, be quite complex to implement in a way that optimizes the efficiency of the GAs search of the problem space. It is this fitness that guides the search of the problem space. Determine the population of ants. Set the intensity of pheromone trial associated with any feature. Determine the maximum of allowed iterations.

3.1.2. Evaluation of the selected subsets

Assign any ant randomly to one feature and visiting features, each ant builds solutions completely. In this step, the evaluation criterion is mean square error (MSE) of the classifier. If an ant is not able to decrease the MSE of the classifier in ten successive steps, it will finish its work and exit. After fitness calculation, the next step is reproduction. Reproduction comprises forming a newpopulation, usually with the sametotal number of chromosomes, by selecting from members of the current population using a stochastic process that is weighted by each of their fitness values [14]. The higher the fitness, the more likely it is that the chromosome will be selected for the new generation.

3.1.3. Construction process

The basic ingredient of anyACOalgorithm is a constructive heuristic for probabilistically constructing solutions [15]. Aconstructi ve heuristic assembles solutions as sequences of elements from the finite set of solution components.

3.1.4. Update process

Decrease pheromone concentrations of nodes then, all ants deposit the quantity of pheromone on graph. Finally, allow the best ant to deposit additional pheromone on nodes.

3.1.5. Decision process

The objective of this module is to find the global best ant so far over all iterations in and to record the selected consequent combination of this ant once it is found in an iteration [16].

3.2. Flow Chart of ACO

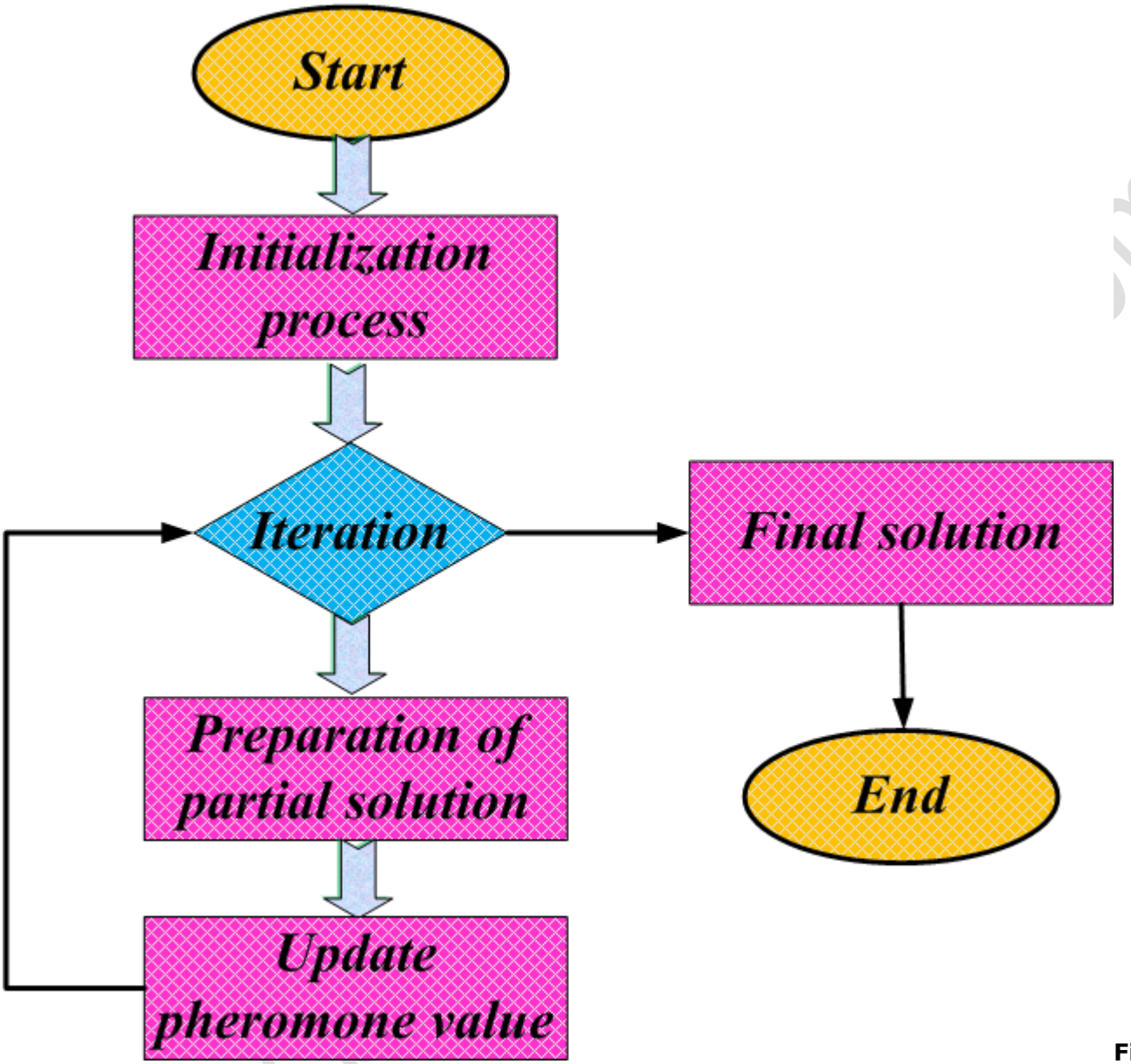


Fig4: Flow Chart of ACO Algorithm

4. Numerical Expression of ACO

The Ant Colony Optimization (ACO) Algorithm Numerical Expressions are given [17];

Minimize $f(X) = 0.6224\, X_1X_3X_4 + 1.7781\, X_2 X_3^2 + 3.1661\, X_1^2 X_4 + 19.84\, X_1^2 X_3$

$f(X) = 0.6224\times2\times4\times5 + 1.7781\times3\times4^2 + 3.1661\times4\times5 + 19.84\times4\times4$

$= 24.896 + 85.3488 + 63.322 + 317.44$

$= 491.0068$

$g_1(X) = - X_1 + 0.0193\, X_3 \le 0,$

$= -2\times0.0193\times4$

$= 0.1544$

$g_2(X) = - X_2 + 0.00954\, X_3 \le 0,$

$= -3\times0.00954\times4$

$= 0.11448$

$g_3(X) = -\pi X_3^2 X_4 - \frac{4}{3}\pi X_3^3 + 1296000 \le 0,$

$= -3.14\times16\times5 - 0.75\times64 + 1296000$

$= 251.2 - 48 + 1296000$

$= 1295796.8$

$g_4(X) = 0.125 - X_1 \le 0$

$= 0.125-2$

$= -1.875$

$g_5(X) = X_1 - X_4 \le 0$

$= 2-5$

$= -3$

$g_6(X) = X_4 - 240 \le 0$

$= 5-240$

$= -235$

$g_7(X) = 0.10471\, X_1^2 + 0.04811\, X_3X_4(14.0 + X_2) - 5.0 \le 0$

$= 0.10471\times2^2 + 0.04811\times4\times5(14.0+3) - 5.0$

$= 0.10471\times4 + 0.04811\times20(17) - 5.0$

$= 0.41884 + 11.3574$

$= 11.77624$

5. Applications of ACO

- Travelling Sales man problem
- Job Shop Scheduling problem[18]
- Network model problem
- Vehicle Routing[19]
- Graph Coloring
- Digital Image Processing
- Quadratic Assignment problem[20]

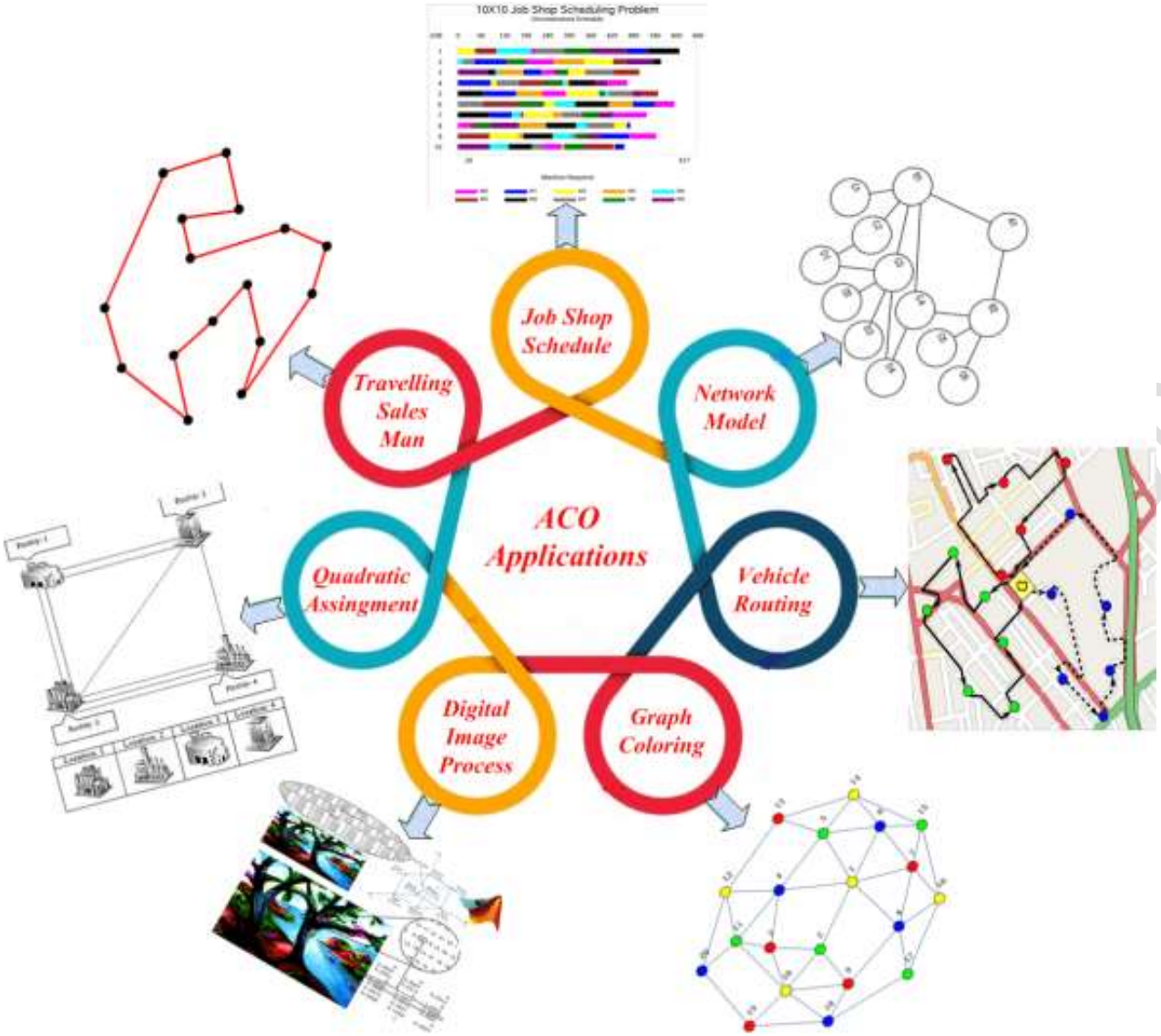


Fig5:

Applications of ACO Algorithm

6. Applications of ACO

- Can be used in dynamic applications
- Positive feedback leads to rapid discovery of good solutions[21]
- Distributed computation avoids premature convergence
- Can search among a population in parallel[22]
- Can adapt to changes such as new distance
- Have guaranteed convergence[23]
- Performance better against other global optimization techniques such as neural net, genetic algorithms, simulated annealing[24]
- The collective interaction of a population of agents
- The greedy heuristic helps find acceptable solution in the early solution in the stages of the search process[25]

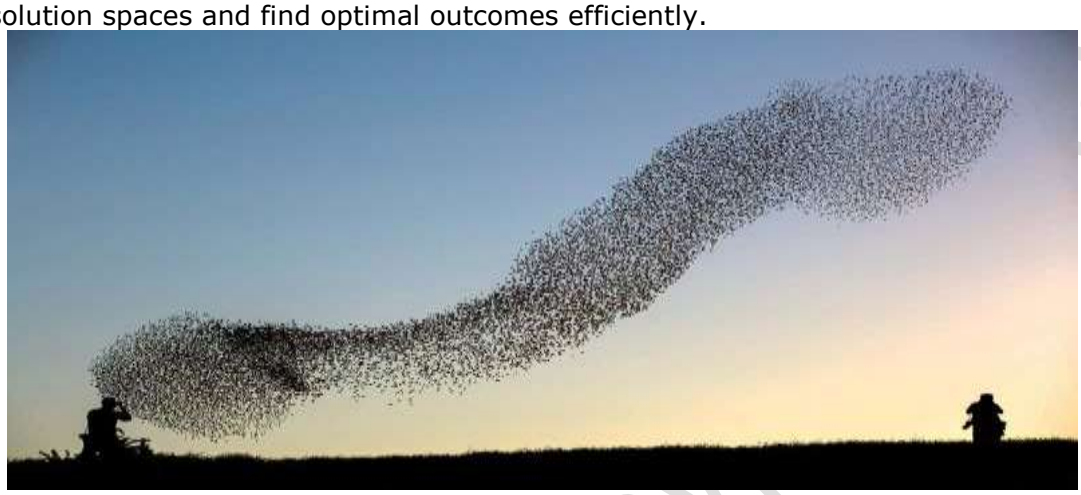
Particle Swarm Optimization

Introduction

Imagine solving complex puzzles by learning from the coordinated movements of birds and fish. Particle Swarm Optimization (PSO) does just that. PSO Algorithm is an intelligent way of solving tricky problems by mimicking how creatures work together. PSO uses many tiny agents that move around to find the best answer. Each agent remembers its own best solution and the best solution from its neighbors. This helps them work together and find the best answer faster. As we explore PSO, we'll uncover how this teamwork between virtual agents helps us crack challenging problems in different areas.

What is PSO Algorithm?

The Particle Swarm Optimization (PSO) algorithm is a computational technique inspired by the collective behavior of natural organisms, such as birds or fish, that move together to achieve a common goal. In PSO, a group of particles (representing potential solutions) navigates through a problem's solution space to find the best possible solution. Each particle adjusts its position based on its own best-known solution (personal best) and the best solution discovered by the entire group (global best). This collaborative movement enables particles to converge toward optimal solutions over iterations. PSO is widely used for optimization problems in various fields, leveraging the power of collective intelligence to explore complex solution spaces and find optimal outcomes efficiently.



Group optimization and Ensemble Learning

Many of you might have encountered the concept of 'No Free Lunch (NFL)' in machine learning. It suggests that no single model is universally superior for all scenarios. In simpler terms, different optimization algorithms can perform equally well when averaged across various problems. To illustrate this idea, consider a flock of birds. Now, why do optimization techniques matter in machine learning and deep learning? When training a model, we define a loss function that quantifies the difference between our model's predictions and actual values. The aim is to minimize or optimize this loss function, driving it closer to zero.

Understanding Ensemble Learning

You might have also encountered the term 'Ensemble Learning.' If not, let me clarify. 'Ensemble' is derived from the French word meaning 'Assembly.' It revolves around learning in a collective or group manner. Imagine training a model using multiple algorithms. What advantages does this offer? A single-base learner is considered weak. However, when these individual learners unite, they become stronger. This amalgamation enhances their predictive power, accuracy, and precision while reducing error rates. This combined model is called 'Meta-learning' in machine learning, where algorithms learn from other algorithms. This approach minimizes variance, bias, and enhances predictive capabilities. Achieving this level of proficiency can be likened to an ultimate 'Nirvana' moment for a data analyst.

The Problem of Optimization

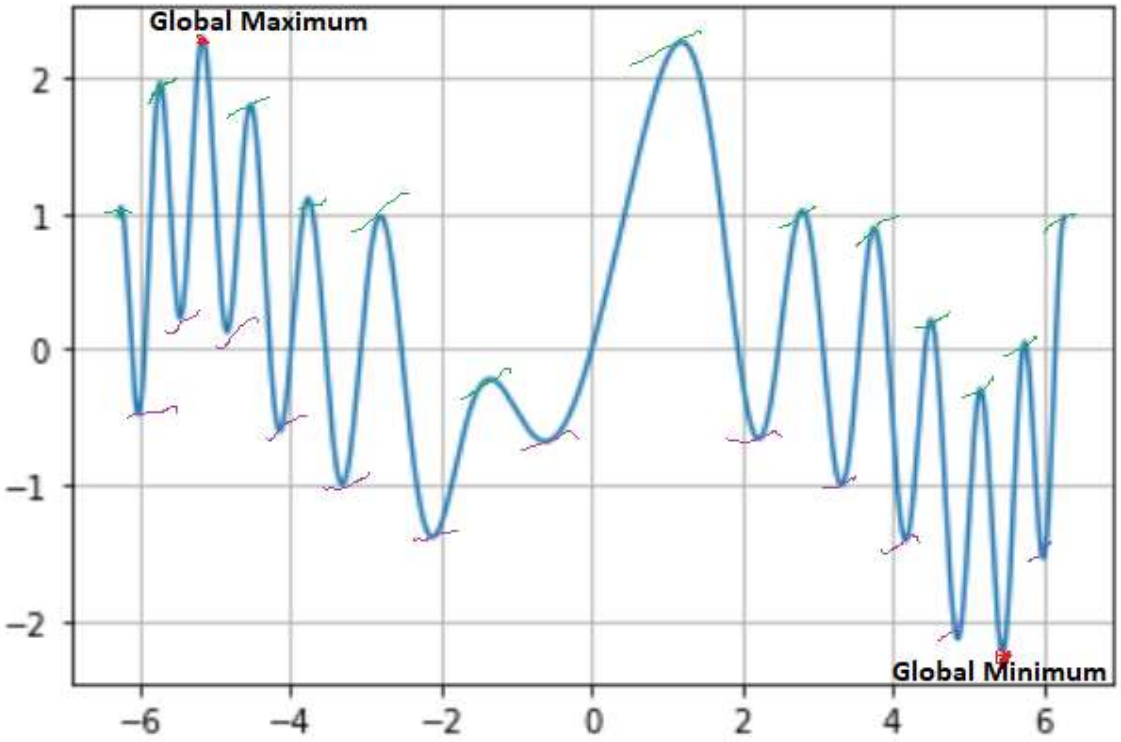
Now let's come back to our PSO model. The concept of swarm intelligence inspired the POS. Here we are speaking about finding the optimal solution in a high-dimensional solution space. It talks about Maximizing earns or minimizing losses. So, we are looking to maximize or minimize a function to find the optimum solution. A function can have multiple local maximum and minimum. But, there can be only one global maximum as well as a minimum. If your function is very complex, then finding the global maximum can be a very daunting task. PSO tries to capture the global maximum or minimum. Even though it cannot capture the exact global maximum/minimum, it goes very close to it. It is the reason we called PSO a heuristic model.

Finding of Global Maximum or Minimum

Let me give you an example of why the finding of global maximum/minimum is problematic. Check the below function :

$$y=f(x)=\sin x+\sin x^2+\sin x\cos x$$

If we plot this function, it looks like below:



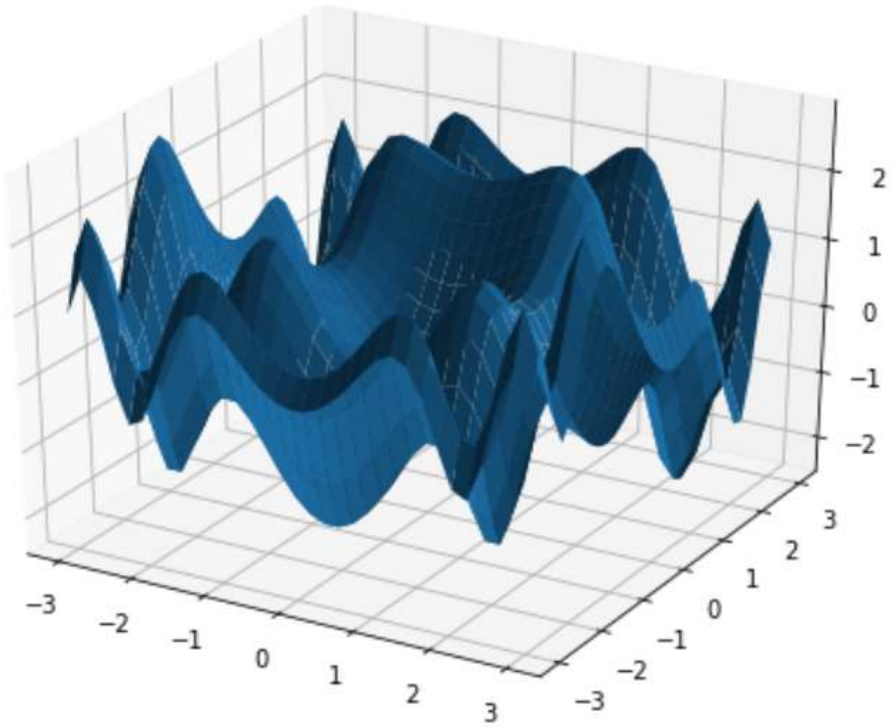
We can see that we have one global maximum and one global minimum. If we consider the function based on an interval in X-axis value from -4 to 6, we will have a maximum that will not be our global maximum. It is a local maximum. So we can say that finding out the global maximum may depend upon the interval. It is something like we try to observe a portion of a continuous function. Also, one thing to note while describing a dynamic system or entity, you can not have a static function. The function that I have defined here is fixed. Data analytics is data-hungry. To train a model or to find a suitable mathematical function, you must have enormous data.

It is impossible to have all the data. Meaning it's challenging to get the exact global minimum or maximum. Well, for me, it's a limitation of Mathematics. Fortunately, we have Statistics that advocate sampling, and from there, it can optimize some value like global maximum or minimum concerning the original function. But again, you won't get the exact global maximum or minimum. You will get some values that will be closer to the actual global maximum or minimum.

PSO Example with Multiple Variables

Also, when we describe a mathematical function based on some real-life scenario, we must explain it with multiple variables or higher-dimensional vector space. The growth of bacteria in a jar may depend upon temperature, humidity, the container, the solvent, etc. For this type of function, it's more challenging to get the exact global maximum and minimum. Check the below function. And see if we add more variables than how difficult it becomes to get global maximum and minimum.

$$z=f(x, y)=\sin x^2+\sin y^2+\sin x\sin y$$



Mathematical Formulation of an Optimization Problem

In the optimization problem, we have a variable represented by a vector $X=[x_1x_2x_3...x_n]$ that minimizes or maximizes cost function depending on the proposed optimization formulation of the function f(X). X is known as **position vector**; it represents a variable model. It is an n dimensions vector, where n represents the number of variables determined in a problem. We can call it **latitude and the longitude** in the problem of choosing a point to land by a flock of birds. The function f(X) is called the **fitness function or objective function**. The job of f(X) is to assess how good or bad a position X is; that is, how perfect a certain landing point a bird thinks after finding a suitable place. Here, the evaluation, in this case, is performed through several survival criteria.

An Intuition of PSO Algorithm

The movement towards a promising area to get the global optimum.

- Each particle adjusts its traveling velocity dynamically, according to the flying experiences it has and its colleagues in the group.
- Each particle tries to keep track of :
  - It's best result for him/her, known as personal best or **pbest**.
  - The best value of any particle is the global best or **gbest**.
- Each particle modifies its position according to:
  - Its current position
  - Its current velocity
  - The distance between its current position and pbest.
  - The distance between its current position and gbest.

Particle Swarm Optimization Algorithm

Lets us assume a few parameters first. You will find some new parameters, which I will describe later.

- f: Objective function
- Vi: Velocity of the particle or agent
- A: Population of agents
- W: Inertia weight
- C1: cognitive constant
- U1, U2: random numbers
- C2: social constant
- Xi: Position of the particle or agent
- P_b: Personal Best
- g_b: global Best

The actual algorithm goes as below :

1. Create a 'population' of agents (particles) which is uniformly distributed over X.
2. Evaluate each particle's position considering the objective function(say the below function).

$$z=f(x, y)=\sin x^2+\sin y^2+\sin x\sin y$$

3. If a particle's present position is better than its previous best position, update it.
4. Find the best particle (according to the particle's last best places).

5. Update particles' velocities.

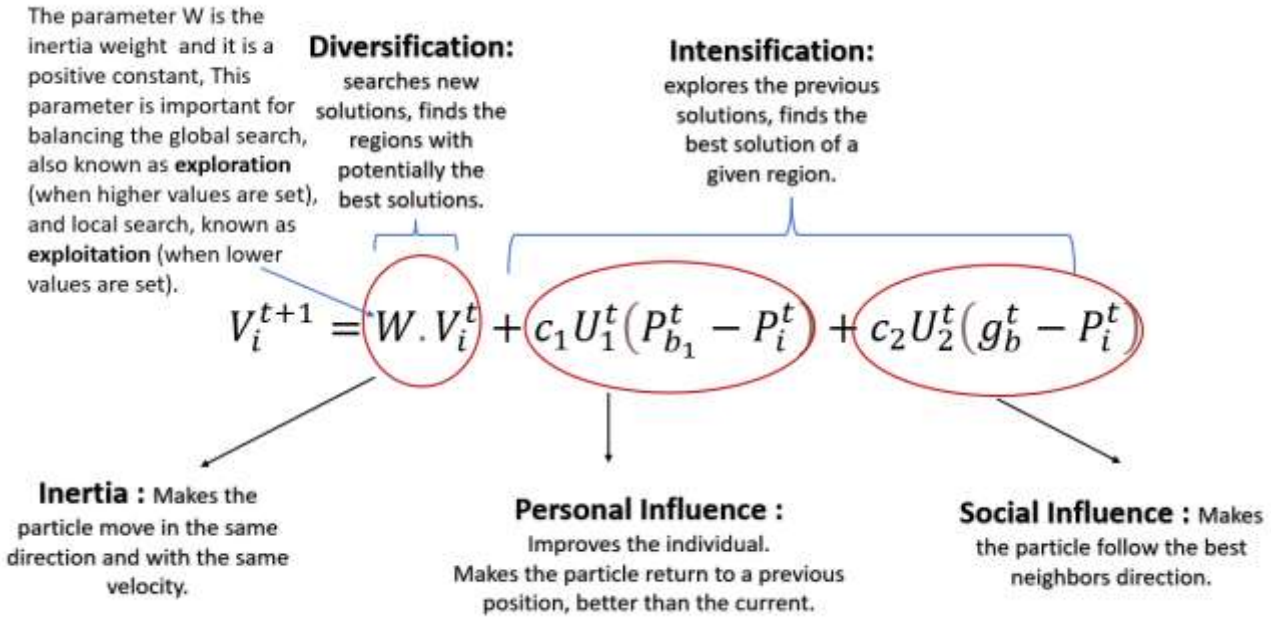
$$V_i^{t+1} = W.V_i^t + c_1 U_1^t\left(P_{b_1}^t - P_i^t\right) + c_2 U_2^t\left(g_b^t - P_i^t\right)$$

6. Move particles to their new positions.

$$P_i^{t+1} = P_i^t + v_i^{t+1}$$

7. Go to step 2 until the stopping criteria are satisfied.

Analysis of the Particle Swarm Optimization Algorithm



If W=1, the particle's motion is entirely influenced by the previous motion, so the particle may keep going in the same direction. On the other hand, if 0≤W<1, such influence is reduced, which means that a particle instead goes to other regions in the search domain.

P_{b1}^t And its current position P_i^t. It has been noticed that the idea behind this term is that as the particle gets more distant from the P_{b1}^t (Personal Best) position, the difference (P_{b1}^t-P_i^t) Must increase; hence, this term increases, attracting the particle to its best own position. The parameter C_1 existing as a product is a positive constant, and it is an individual-cognition parameter. It weighs the importance of the particle's own previous experiences.

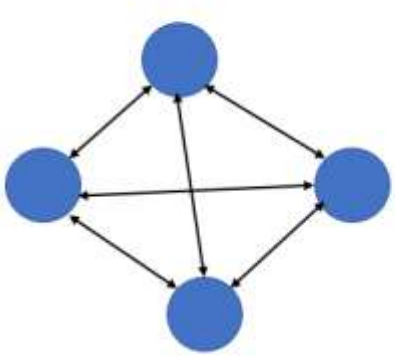
The other hyper-parameter which composes the product of the second term is U_1^t. It is a random value parameter with [0,1] range. This random parameter plays an essential role in avoiding premature convergences, increasing the most likely global optima.

The difference (g_b^t-P_i^t) Works as an attraction for the particles towards the best point until it's found at t iteration. Likewise, C_2 is also a social learning parameter, and it weighs the importance of the global learning of the swarm. And U_2^t plays precisely the same role as U_1^t.

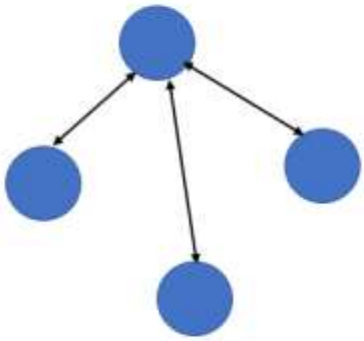
- C_1=C_2=0, all particles continue flying at their current speed until they hit the search space's boundary.
- C_1>0 and C_2=0, all particles are independent.
- C_1>0 and C_2=0, all particles are attracted to a single point in the entire swarm.
- C_1=C_2≠0, all particles are attracted towards the average of pbest and gbest.

Neighborhood Topologies

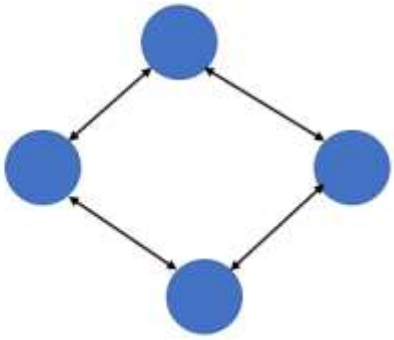
A neighborhood must be defined for each particle. This neighborhood determines the extent of social interaction within the swarm and influences a particular particle's movement. Less interaction occurs when the neighborhoods in the swarm are small. For small neighborhoods, the convergence will be slower, but it may improve the quality of solutions. The convergence will be faster for more prominent neighborhoods, but the risk that sometimes convergence occurs earlier.



Star Topology



Wheel Topology



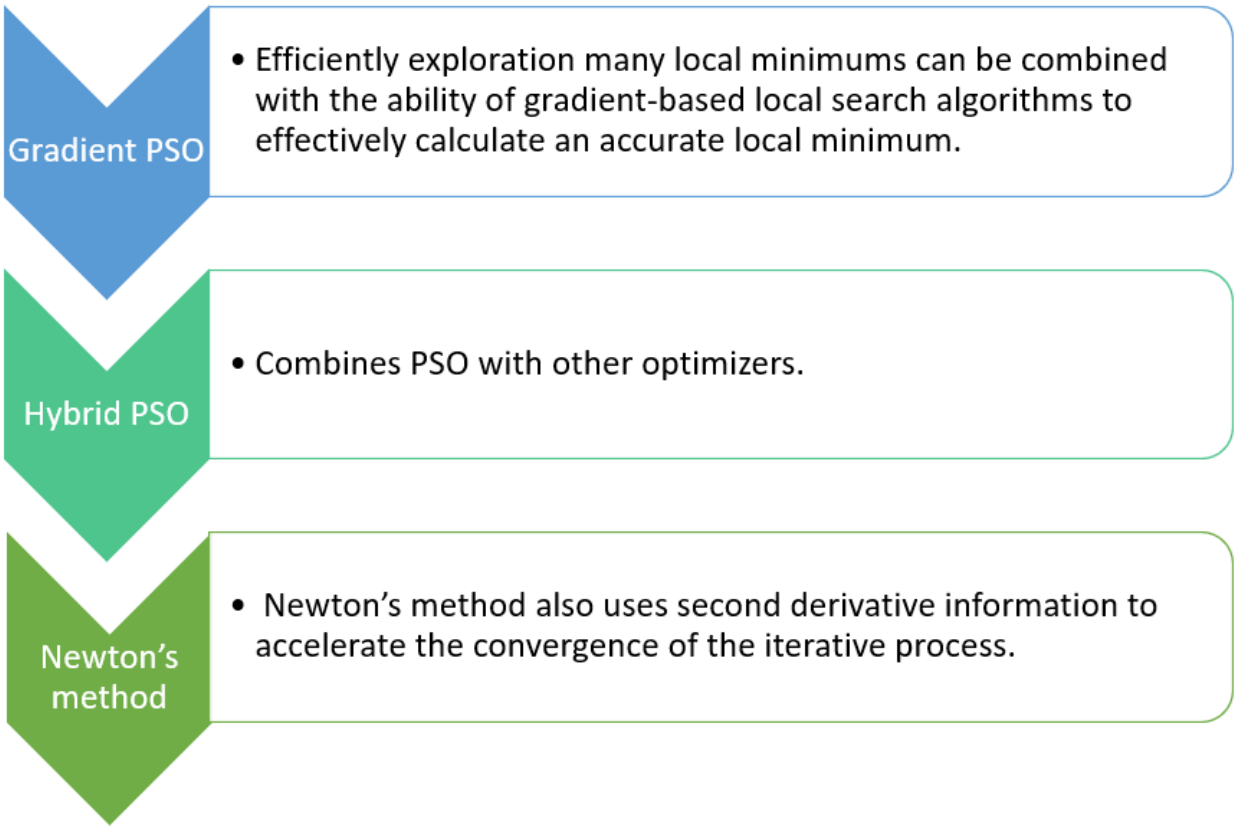
Ring Topology

For Star topology, each particle is connected with other particles. It leads to faster convergence than other topologies, Easy to find out gbest. But it can be biased to the pbest.

For wheel topology, only one particle connects to the others, and all information is communicated through this particle. This focal particle compares the best performance of all particles in the swarm, and adjusts its position towards the best performance particle. Then the new position of the focal particle is informed to all the particles.

For Ring Topology, when one particle finds the best result, it will make pass it to its immediate neighbors, and these two immediate neighbors pass it to their immediate neighbors until it reaches the last particle. Here the best result found is spread very slowly.

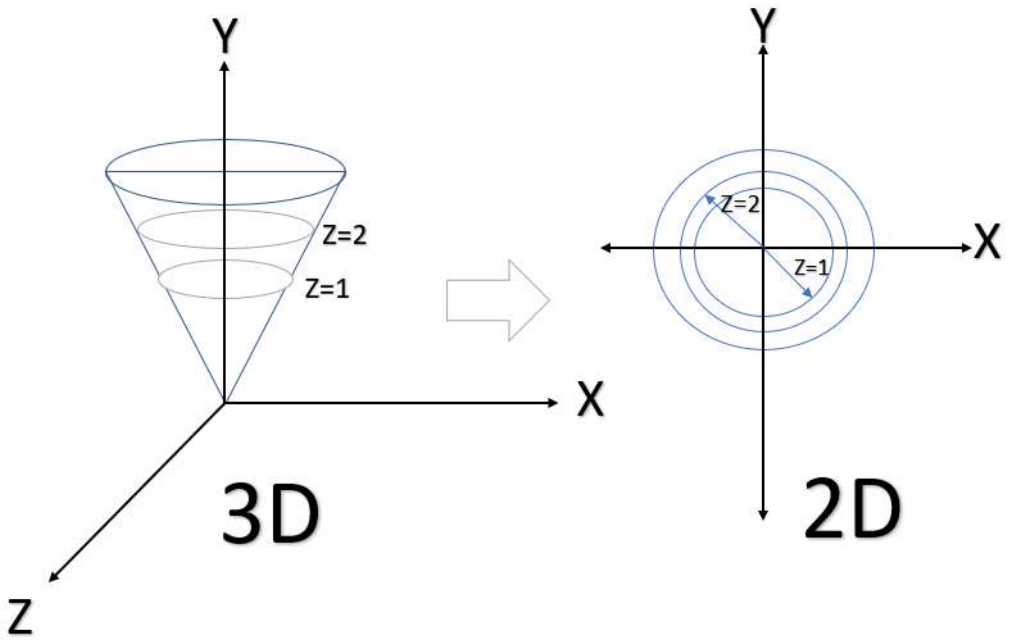
Types of Particle Swarm Optimization

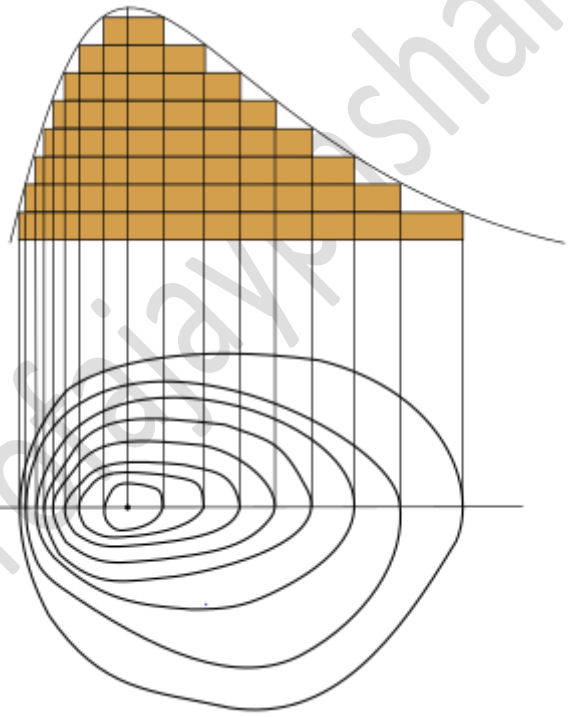


Contour Plot

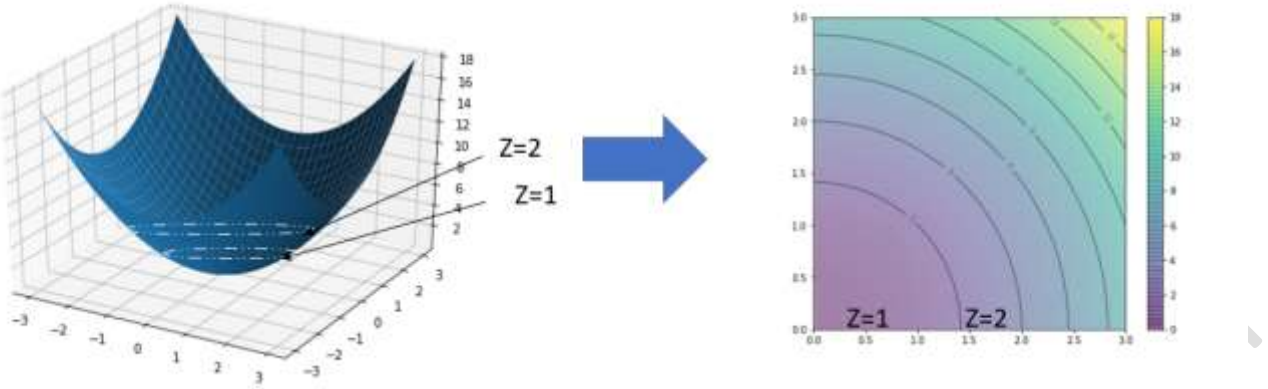
It is a graphical technique to represent 3 -Dimensional surface in 2- dimensional Plot using variable Z in the form of slices known as contours. I hope the below example can give you the intuition.

Let's draw a graph of circle $z=x^2+y^2$ at fixed heights 'z' , z =1,2,3 etc.

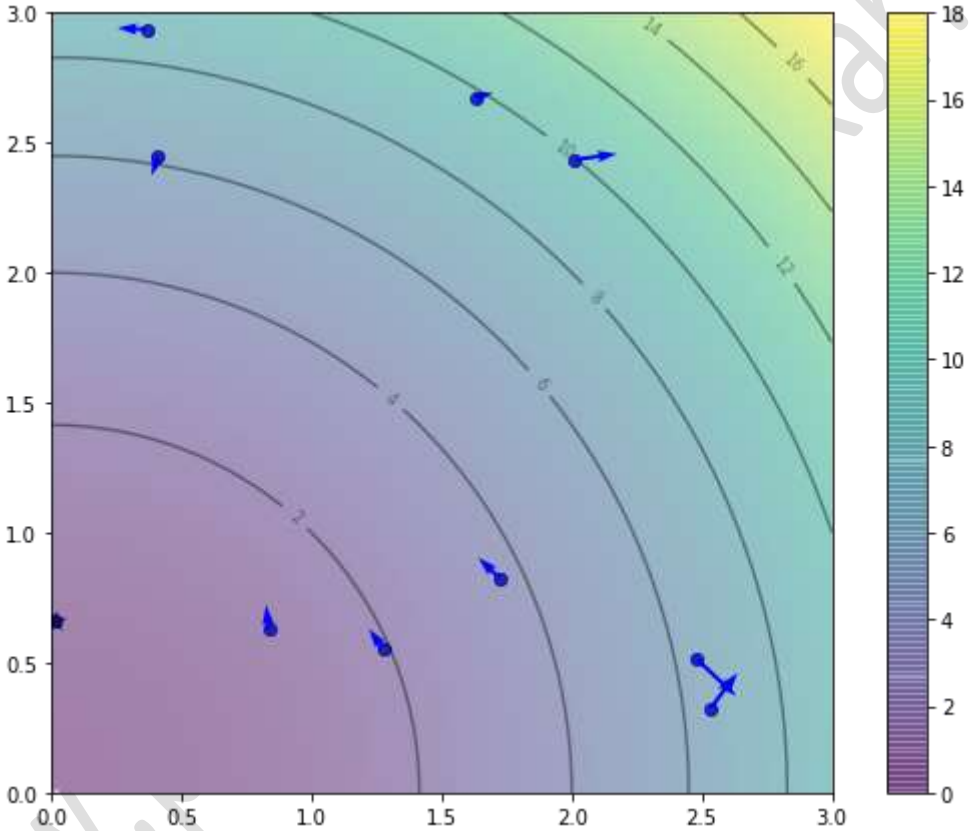




To give you intuition, let Plot the function below in the contour plot.
$z=x^2+y^2$ its actual plotting and the contour plotting will look like below:



Here we can see the function in the region of f(x,y). We can create ten particles at random locations in this region, together with a random velocity which is sampled over a normal distribution with mean 0 and standard deviation 0.1, as follows:



The actual outcome will be like :

PSO found best solution at f([0.01415657 0.65909248])=0.4346033028251361

Global optimal at f([0.0, 0.0])=0.0

For details coding part, I'll highly recommend you to visit the link: **https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/**

Also, there is a library available called as pyswarms; please check here to know more!

Difference between PSO and Genetic Algorithm

Genetic Algorithms (GAs) and PSOs are both used as cost functions, they are both iterative, and they both have a random element. They can be used on similar kinds of problems. The difference between PSO and Genetic Algorithms (GAs) is that GAs it does not traverse the search space like birds flocking, covering the spaces in between. The operation of GAs is more like Monte Carlo, where the candidate solutions are randomized, and the best solutions are picked to compete with a new set of randomized solutions. Also, PSO algorithms require normalization of the input vectors to reach faster "convergence" (as heuristic algorithms, both don't truly converge). GAs can work with features that are continuous or discrete.

Also, In PSO, there is no creation or deletion of individuals. Individuals merely move on a landscape where their fitness is measured over time. This is like a flock of birds or other creatures that communicate.

Advantages and disadvantages of Particle Swarm Optimization

Advantages :

1. Insensitive to scaling of design variables.
2. Easily parallelized for concurrent processing.
3. Derivative free.
4. Very few algorithm parameters.
5. A very efficient global search algorithm.

Disadvantages :

1. PSO's optimum local searchability is weak

Conclusion

The most exciting part of PSO Algorithm is there is a stable topology where particles are able to communicate with each other and increase the learning rate to achieve global optimum. The metaheuristic nature of this optimization algorithm gives us lots of opportunities as it optimizes a problem by iteratively trying to improve a candidate solution. Applicability of it will increase more with the ongoing research work in Ensemble Learning.

Frequently Asked Question

Q1. What is the principle of PSO algorithm?

A. Particle Swarm Optimization (PSO) simulates the social behavior of birds or fish, where particles (solutions) move through a solution space, adjusting their positions based on their own best-known solution and the collective knowledge of the swarm.

Q2. Why is PSO algorithm used?

A. PSO is used to optimize complex problems by iteratively improving solutions based on particle movement and their interactions, making it effective for optimization tasks in various fields.

Q3. Who developed PSO algorithm?

A. PSO was introduced by Dr. Eberhart and Dr. Kennedy in 1995, inspired by the social behaviors of birds and fish.

Q4. Where is PSO algorithm used?

A. PSO finds applications in diverse fields, including engineering design, neural network training, economic modeling, data clustering, and parameter tuning in machine learning algorithms.

CHAPTER VIII: INTELLIGENT AGENTS

Topics Covered:
Agents vs software programs, classification of agents, working of an agent, single agent and multiagent systems, performance evaluation, architecture, agent communication language, applications

Agents in Artificial Intelligence

An AI system can be defined as the study of the rational agent and its environment. The agents sense the environment through sensors and act on their environment through actuators. An AI agent can have mental properties such as knowledge, belief, intention, etc.

What is an Agent?

An agent can be anything that perceiveits environment through sensors and act upon that environment through actuators. An Agent runs in the cycle of **perceiving**, **thinking**, and **acting**. An agent can be:

- **Human-Agent:** A human agent has eyes, ears, and other organs which work for sensors and hand, legs, vocal tract work for actuators.
- **Robotic Agent:** A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.
- **Software Agent:** Software agent can have keystrokes, file contents as sensory input and act on those inputs and display output on the screen.

Hence the world around us is full of agents such as thermostat, cellphone, camera, and even we are also agents.

Before moving forward, we should first know about sensors, effectors, and actuators.

Sensor: Sensor is a device which detects the change in the environment and sends the information to other electronic devices. An agent observes its environment through sensors.

Actuators: Actuators are the component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system. An actuator can be an electric motor, gears, rails, etc.

Effectors: Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers, wings, fins, and display screen.

Intelligent Agents:

An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals. An intelligent agent may learn from the environment to achieve their goals. A thermostat is an example of an intelligent agent.

Following are the main four rules for an AI agent:

- **Rule 1:** An AI agent must have the ability to perceive the environment.
- **Rule 2:** The observation must be used to make decisions.
- **Rule 3:** Decision should result in an action.
- **Rule 4:** The action taken by an AI agent must be a rational action.

Rational Agent:
A rational agent is an agent which has clear preference, models uncertainty, and acts in a way to maximize its performance measure with all possible actions.

A rational agent is said to perform the right things. AI is about creating rational agents to use for game theory and decision theory for various real-world scenarios.

For an AI agent, the rational action is most important because in AI reinforcement learning algorithm, for each best possible action, agent gets the positive reward and for each wrong action, an agent gets a negative reward.

Note: Rational agents in AI are very similar to intelligent agents.

Rationality:

The rationality of an agent is measured by its performance measure. Rationality can be judged on the basis of following points:

- Performance measure which defines the success criterion.
- Agent prior knowledge of its environment.
- Best possible actions that an agent can perform.
- The sequence of percepts.

Note: Rationality differs from Omniscience because an Omniscient agent knows the actual outcome of its action and act accordingly, which is not possible in reality.

Structure of an AI Agent

The task of AI is to design an agent program which implements the agent function. The structure of an intelligent agent is a combination of architecture and agent program. It can be viewed as:

1. Agent = Architecture + Agent program

Following are the main three terms involved in the structure of an AI agent:

Architecture: Architecture is machinery that an AI agent executes on.

Agent Function: Agent function is used to map a percept to an action.

1. $f:P^* \rightarrow A$

Agent program: Agent program is an implementation of agent function. An agent program executes on the physical architecture to produce function f.

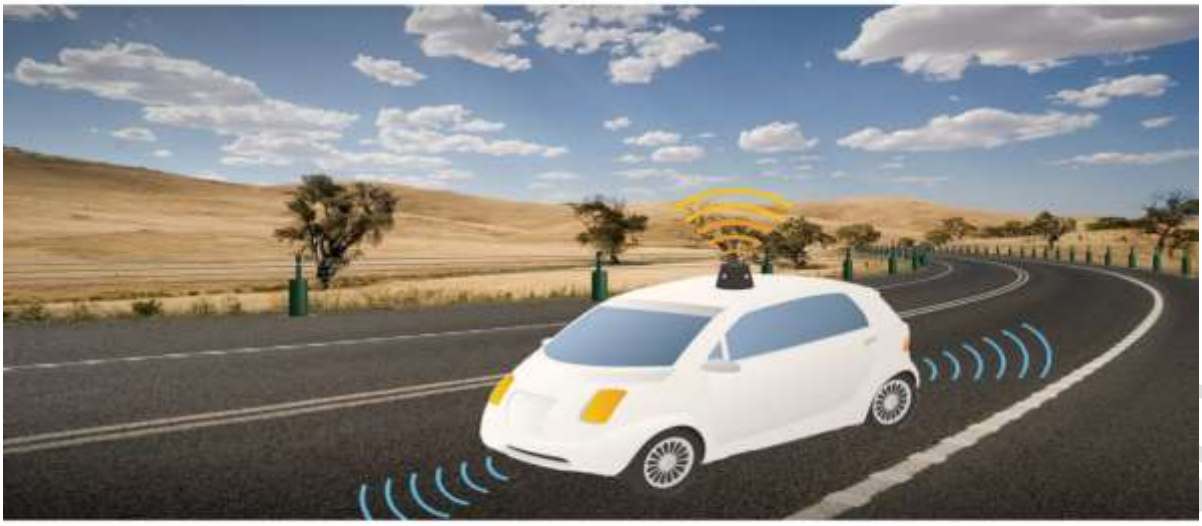
PEAS Representation

PEAS is a type of model on which an AI agent works upon. When we define an AI agent or rational agent, then we can group its properties under PEAS representation model. It is made up of four words:

- **P:** Performance measure
- **E:** Environment
- **A:** Actuators
- **S:** Sensors

Here performance measure is the objective for the success of an agent's behavior.

PEAS for self-driving cars:



Let's suppose a self-driving car then PEAS representation will be:

Performance: Safety, time, legal drive, comfort

Environment: Roads, other vehicles, road signs, pedestrian

Actuators: Steering, accelerator, brake, signal, horn

Sensors: Camera, GPS, speedometer, odometer, accelerometer, sonar.

Example of Agents with their PEAS representation

Agent	Performance measure	Environment	Actuators	Sensors
1. Medical Diagnose	○ Healthy patient ○ Minimized cost	○ Patient ○ Hospital ○ Staff	○ Tests ○ Treatments	Keyboard (Entry of symptoms)
2. Vacuum Cleaner	○ Cleanness ○ Efficiency ○ Battery life ○ Security	○ Room ○ Table ○ Wood floor ○ Carpet ○ Various obstacles	○ Wheels ○ Brushes ○ Vacuum Extractor	○ Camera ○ Dirt detection sensor ○ Cliff sensor ○ Bump Sensor ○ Infrared Wall

				Sensor
3. Part - picking Robot	○ Percentage of parts in correct bins.	○ Conveyor belt with parts, ○ Bins	○ Jointed Arms ○ Hand	○ Camera ○ Joint angle sensors.

Uses of Agents

Agents are used in a wide range of applications in artificial intelligence, including:

- **Robotics:** Agents can be used to control robots and automate tasks in manufacturing, transportation, and other industries.
- **Smart homes and buildings:** Agents can be used to control heating, lighting, and other systems in smart homes and buildings, optimizing energy use and improving comfort.
- **Transportation systems:** Agents can be used to manage traffic flow, optimize routes for autonomous vehicles, and improve logistics and supply chain management.
- **Healthcare:** Agents can be used to monitor patients, provide personalized treatment plans, and optimize healthcare resource allocation.
- **Finance:** Agents can be used for automated trading, fraud detection, and risk management in the financial industry.
- **Games:** Agents can be used to create intelligent opponents in games and simulations, providing a more challenging and realistic experience for players.
- **Natural language processing:** Agents can be used for language translation, question answering, and chatbots that can communicate with users in natural language.
- **Cybersecurity:** Agents can be used for intrusion detection, malware analysis, and network security.
- **Environmental monitoring:** Agents can be used to monitor and manage natural resources, track climate change, and improve environmental sustainability.
- **Social media:** Agents can be used to analyze social media data, identify trends and patterns, and provide personalized recommendations to users.

Agent Environment in AI

An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself. An environment can be described as a situation in which an agent is present.

The environment is where agent lives, operate and provide the agent with something to sense and act upon it. An environment is mostly said to be non-feministic.

Features of Environment

As per Russell and Norvig, an environment can have various features from the point of view of an agent:

1. Fully observable vs Partially Observable
2. Static vs Dynamic
3. Discrete vs Continuous
4. Deterministic vs Stochastic
5. Single-agent vs Multi-agent
6. Episodic vs sequential
7. Known vs Unknown
8. Accessible vs Inaccessible

1**. Fully observable vs Partially Observable**:

- If an agent sensor can sense or access the complete state of an environment at each point of time then it is **a fully observable** environment, else it is **partially observable**.
- A fully observable environment is easy as there is no need to maintain the internal state to keep track history of the world.
- An agent with no sensors in all environments then such an environment is called as **unobservable**.

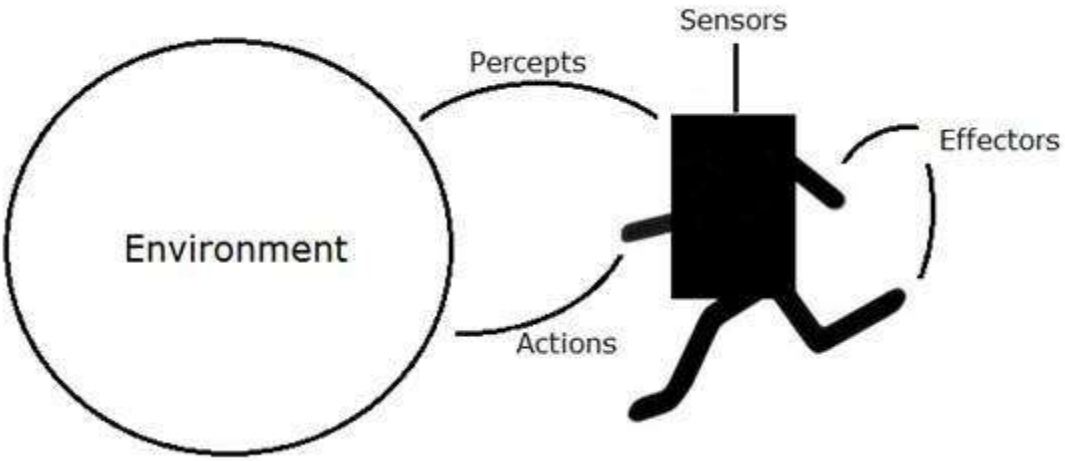
2. **Deterministic vs Stochastic**:

- If an agent's current state and selected action can completely determine the next state of the environment, then such environment is called a deterministic environment.
- A stochastic environment is random in nature and cannot be determined completely by an agent.
- In a deterministic, fully observable environment, agent does not need to worry about uncertainty.

3. **Episodic vs Sequential:**
   - In an episodic environment, there is a series of one-shot actions, and only the current percept is required for the action.
   - However, in Sequential environment, an agent requires memory of past actions to determine the next best actions.
4. **Single-agent vs Multi-agent**
   - If only one agent is involved in an environment, and operating by itself then such an environment is called single agent environment.
   - However, if multiple agents are operating in an environment, then such an environment is called a multi-agent environment.
   - The agent design problems in the multi-agent environment are different from single agent environment.
5. **Static vs Dynamic**:
   - If the environment can change itself while an agent is deliberating then such environment is called a dynamic environment else it is called a static environment.
   - Static environments are easy to deal because an agent does not need to continue looking at the world while deciding for an action.
   - However for dynamic environment, agents need to keep looking at the world at each action.
   - Taxi driving is an example of a dynamic environment whereas Crossword puzzles are an example of a static environment.
6. **Discrete vs Continuous:**
   - If in an environment there are a finite number of percepts and actions that can be performed within it, then such an environment is called a discrete environment else it is called continuous environment.
   - A chess gamecomes under discrete environment as there is a finite number of moves that can be performed.
   - A self-driving car is an example of a continuous environment.
7. **Known vs Unknown**
   - Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.
   - In a known environment, the results for all actions are known to the agent. While in unknown environment, agent needs to learn how it works in order to perform an action.
   - It is quite possible that a known environment to be partially observable and an Unknown environment to be fully observable.
8. **Accessible vs Inaccessible**
   - If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible.
   - An empty room whose state can be defined by its temperature is an example of an accessible environment.
   - Information about an event on earth is an example of Inaccessible environment.

An AI system is composed of an agent and its environment. The agents act in their environment. The environment may contain other agents.
What are Agent and Environment?
An **agent** is anything that can perceive its environment through **sensors** and acts upon that environment through **effectors.**

- A **human agent** has sensory organs such as eyes, ears, nose, tongue and skin parallel to the sensors, and other organs such as hands, legs, mouth, for effectors.
- A **robotic agent** replaces cameras and infrared range finders for the sensors, and various motors and actuators for effectors.
- A **software agent** has encoded bit strings as its programs and actions.



Agent Terminology

- **Performance Measure of Agent** – It is the criteria, which determines how successful an agent is.
- **Behavior of Agent** – It is the action that agent performs after any given sequence of percepts.
- **Percept** – It is agent's perceptual inputs at a given instance.
- **Percept Sequence** – It is the history of all that an agent has perceived till date.
- **Agent Function** – It is a map from the precept sequence to an action.

Rationality

Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment.

Rationality is concerned with expected actions and results depending upon what the agent has perceived. Performing actions with the aim of obtaining useful information is an important part of rationality.

What is Ideal Rational Agent?

An ideal rational agent is the one, which is capable of doing expected actions to maximize its performance measure, on the basis of –

- Its percept sequence
- Its built-in knowledge base

Rationality of an agent depends on the following –

- The **performance measures**, which determine the degree of success.
- Agent's **Percept Sequence** till now.
- The agent's **prior knowledge about the environment**.
- The **actions** that the agent can carry out.

A rational agent always performs right action, where the right action means the action that causes the agent to be most successful in the given percept sequence. The problem the agent solves is characterized by Performance Measure, Environment, Actuators, and Sensors (PEAS).

The Structure of Intelligent Agents

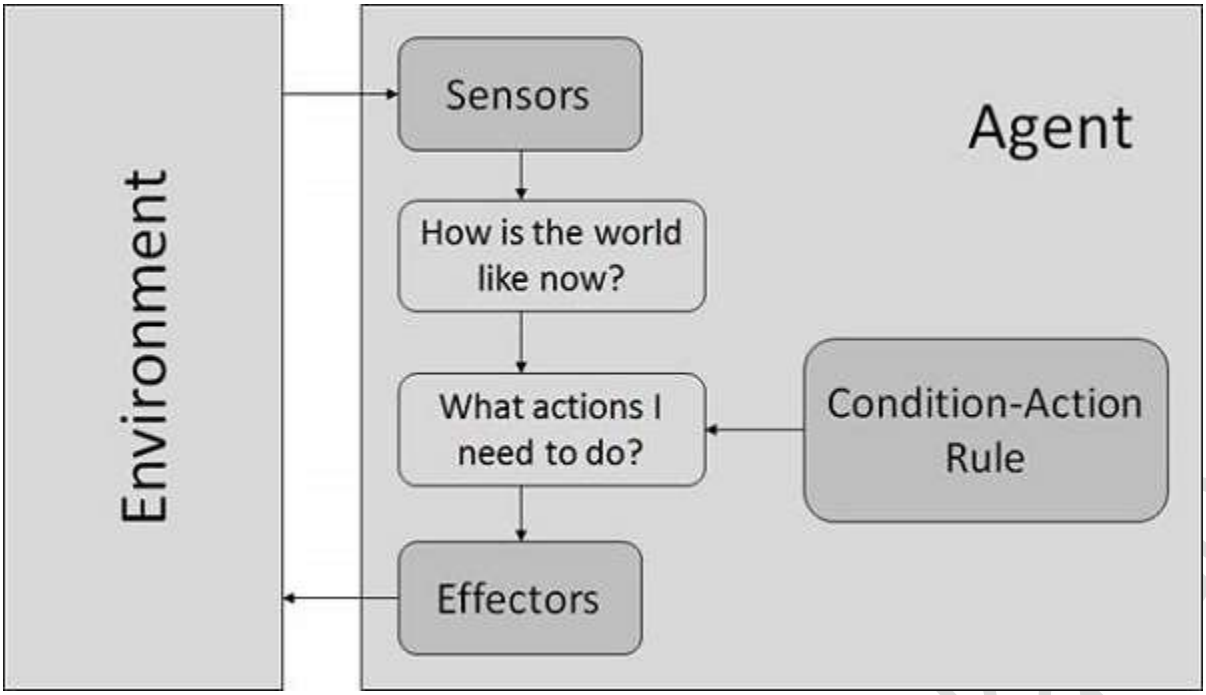
Agent's structure can be viewed as –

- Agent = Architecture + Agent Program
- Architecture = the machinery that an agent executes on.
- Agent Program = an implementation of an agent function.

Simple Reflex Agents

- They choose actions only based on the current percept.
- They are rational only if a correct decision is made only on the basis of current precept.
- Their environment is completely observable.

Condition-Action Rule – It is a rule that maps a state (condition) to an action.



Model Based Reflex Agents

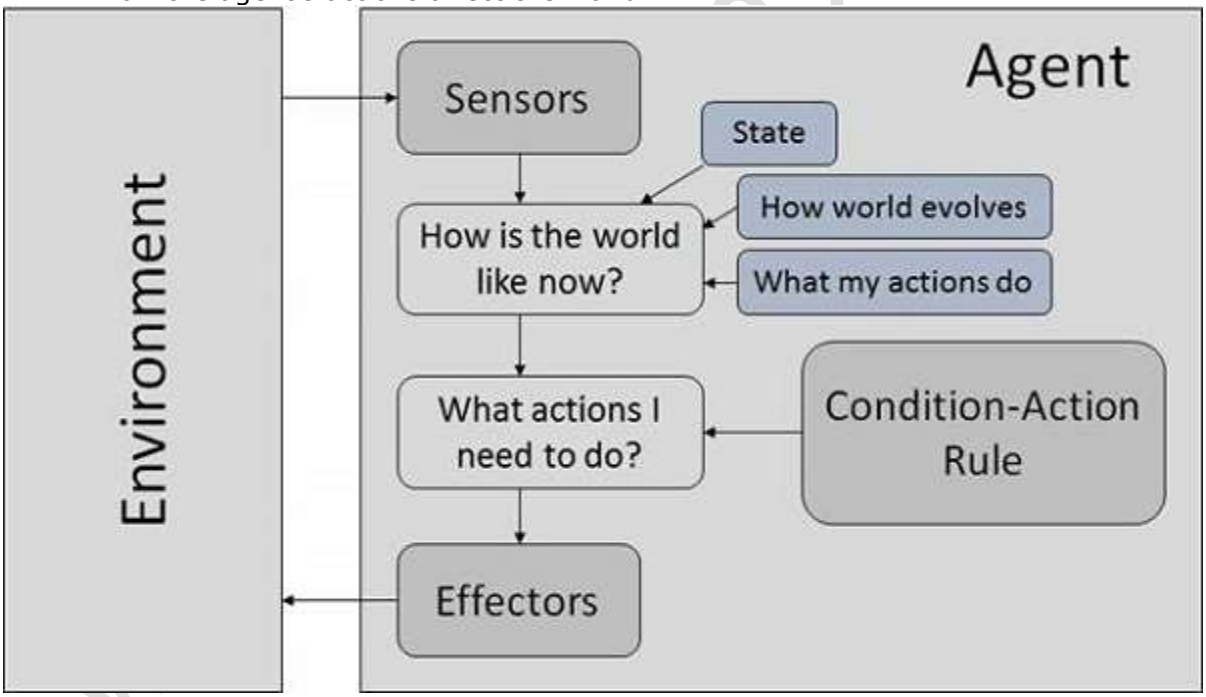
They use a model of the world to choose their actions. They maintain an internal state.

Model – knowledge about "how the things happen in the world".

Internal State – It is a representation of unobserved aspects of current state depending on percept history.

Updating the state requires the information about –

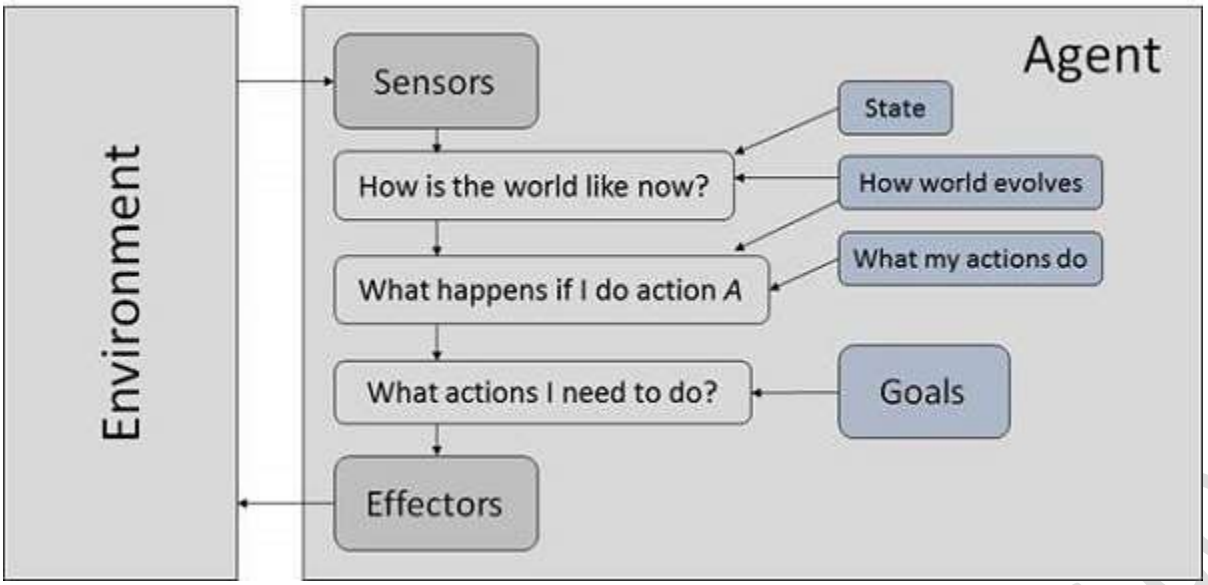
- How the world evolves.
- How the agent's actions affect the world.



Goal Based Agents

They choose their actions in order to achieve goals. Goal-based approach is more flexible than reflex agent since the knowledge supporting a decision is explicitly modeled, thereby allowing for modifications.

Goal – It is the description of desirable situations.

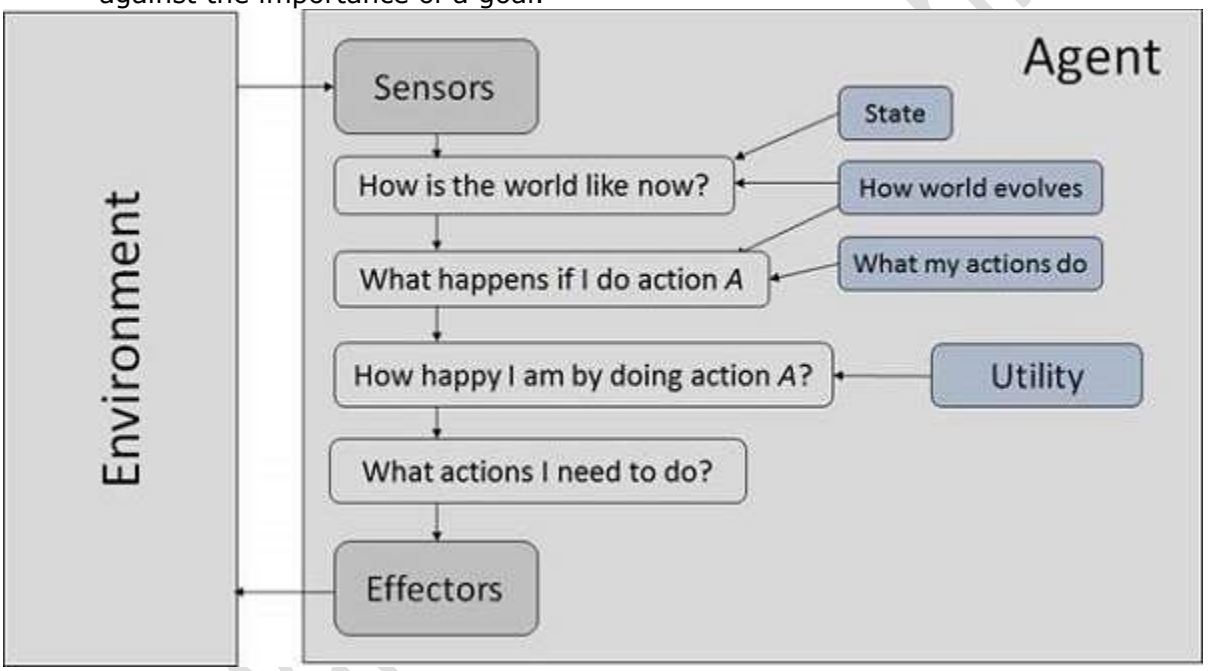


Utility Based Agents

They choose actions based on a preference (utility) for each state.

Goals are inadequate when –

- There are conflicting goals, out of which only few can be achieved.
- Goals have some uncertainty of being achieved and you need to weigh likelihood of success against the importance of a goal.



The Nature of Environments

Some programs operate in the entirely **artificial environment** confined to keyboard input, database, computer file systems and character output on a screen.

In contrast, some software agents (software robots or softbots) exist in rich, unlimited softbots domains. The simulator has a **very detailed, complex environment**. The software agent needs to choose from a long array of actions in real time. A softbot designed to scan the online preferences of the customer and show interesting items to the customer works in the **real** as well as an **artificial** environment.

The most famous **artificial environment** is the **Turing Test environment**, in which one real and other artificial agents are tested on equal ground. This is a very challenging environment as it is highly difficult for a software agent to perform as well as a human.

Turing Test

The success of an intelligent behavior of a system can be measured with Turing Test.

Two persons and a machine to be evaluated participate in the test. Out of the two persons, one plays the role of the tester. Each of them sits in different rooms. The tester is unaware of who is machine and who is a human. He interrogates the questions by typing and sending them to both intelligences, to which he receives typed responses.

This test aims at fooling the tester. If the tester fails to determine machine's response from the human response, then the machine is said to be intelligent.

Properties of Environment
The environment has multifold properties –

- **Discrete / Continuous** – If there are a limited number of distinct, clearly defined, states of the environment, the environment is discrete (For example, chess); otherwise it is continuous (For example, driving).
- **Observable / Partially Observable** – If it is possible to determine the complete state of the environment at each time point from the percepts it is observable; otherwise it is only partially observable.
- **Static / Dynamic** – If the environment does not change while an agent is acting, then it is static; otherwise it is dynamic.
- **Single agent / Multiple agents** – The environment may contain other agents which may be of the same or different kind as that of the agent.
- **Accessible / Inaccessible** – If the agent's sensory apparatus can have access to the complete state of the environment, then the environment is accessible to that agent.
- **Deterministic / Non-deterministic** – If the next state of the environment is completely determined by the current state and the actions of the agent, then the environment is deterministic; otherwise it is non-deterministic.
- **Episodic / Non-episodic** – In an episodic environment, each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself. Subsequent episodes do not depend on the actions in the previous episodes. Episodic environments are much simpler because the agent does not need to think ahead

Why is communication important?
•Most, but not all, would agree that communication is a requirement for cooperation.
•Societies can do things that no individual (agent) can.
•Diversity introduces heterogeneity.
•Autonomy encourages disregard for other agents' internal structure.
•Communicating agents need only care about understanding a "common language".

Agent Communication
•Agent-to-agent communication is key to realizing the potential of the agent paradigm, just as the development of human language was key to the development of human intelligence and societies.
•Agents use an Agent Communication Language or ACL to communication information and knowledge.
•Genesereth (CACM, 1992) defined a software agent as any system which uses an ACL to exchange information.

Communication and knowledge-sharing characteristics are necessary in order to achieve interoperability in **Multi-Agent System**. Knowledge has to be represented in a meaningful way for agents to be able to understand the information transmitted between them. In the previous article, we looked into the concept of **knowledge representation** in Artificial Intelligence and mentioned some popular examples of languages used for defining ontology and the body of knowledge including Knowledge Interchange Format (KIF) and Extensible Mark-up Language (XML). In this article, we will look into some of the Agent Communication Languages (ACL) in AI that have been developed to standardize the way agents communicate, including KQML, FIPA ACL and SOAP.

Agent Communication Languages and Protocols for Interoperability of Intelligent Agents

The interoperability of **Intelligent Agents** is an important issue in **Multi-Agent System**. The ability to communicate and share knowledge is necessary for agents to interact. The interaction and cooperation of agents are impossible without standardized ways of communicating knowledge. Standardize communication protocols are required to ensure that agents have a standard way of communicating with each other.

Different strategies and protocols have been developed to enable communication between software components and the exchange of information between applications [1]. These communication protocols include Remote Procedure Call (RPC), Remote Method Invocation (RMI) in Java, Distributed Component Object Model (DCOM), and Common Object Request Broker Architecture (CORBA). CORBA is a distributed object protocol created by the Object Management Group (OMG) to enable object-level interoperability [2]. OMG is a leader in defining specifications in the area of mobile agent systems. CORBA enables the communication between software components and procedures that are written in different languages on different computers. Its messages contain procedures but no semantics.

As discussed earlier, **Intelligent Agents** are autonomous and computational entities that can control internal states and actions, and make decisions on how to reply or react to messages from other

agents. Such communication protocols allow agents to directly transfer information and change the state or behavior of other agents, but this is insufficient for total inter-agent expression [3, 4]. These protocols are a viable means of conveying messages between agents, but higher-level communication protocols with rich semantics would be required to enable higher-level agent interactions including cooperation and negotiation.

ACLs are distinguished from other communication methods (e.g. CORBA) by their semantic complexity. They allow handling of complex semantics and exchange of more complex objects including agents' desired states, goals, experience, and shared plans. Agents using ACLs are able to engage in task-oriented conversations, which include shared sequences of messages transmitted between them.

ACLs are high-level communication methods and use lower-level communication protocols for transferring information. Examples of lower-level communication protocols include TCP/IP, Simple Mail Transfer Protocol (SMTP), and HyperText Transfer Protocol (HTTP) to transfer information over a network. A number of ACLs have been developed for this purpose including KQML, FIPA ACL, and SOAP.

The following sections discuss these ACLs in more detail and describe the *Speech Act* theory, which has inspired and influenced the creation of these languages.

Speech Act Theory

The research on ACLs was inspired by the work of philosopher John Austin and his *Speech Act* theory [5]. Austin stated that a certain class of utterances in communication could be considered as actions, which can change the state of the world. These utterances are referred to as *speech acts* or *performative verbs*. Speech acts can represent the state and intention of agents, or affect their plans and understanding of the environment.

Searle [6] continued Austin's work in 1969 and consequently formalized the structure of speech acts by defining their different classes and the necessary and sufficient conditions for their successful performance. Cohen and Levesque [7, 8] introduced speech acts in AI and developed a plan-based theory of speech acts. "They showed how pre-and post-conditions of speech acts could be represented in a multi-modal logic" [9].

There are many types of speech acts. Perhaps the most recognised ones are representatives (that provide some information) and directive (that request something). Some examples of these include; *'request'*, *'inform'*, *'suggest'*, *'query'*, and *'promise'*.

The Speech Act theory has influenced a number of ACL and knowledge-sharing techniques. These languages have been developed to enable representation and exchange of knowledge between agents [10]. In the following sections some of these languages are presented.

Knowledge Query Manipulation Language

"Knowledge Query Manipulation Language (KQML) is an ACL developed as part of the DARPA Knowledge-Sharing Effort (KSE)" [11]. It is a high-level, message-oriented communication language developed and based on Speech Act theory [10]. KQML focuses on message format and message-handling protocols. It is independent of transport mechanisms, content-language, and ontology. Some examples of transport mechanisms include: TCP/IP, HTTP, and SMTP.

KQML messages consist of three layers: message, content, and communication layer [12, 13]. The message layer is the core of KQML, which contains performatives and parameters for identifying network protocols to deliver the message. Performatives describe the intention of an agent to send the message. The content layer contains the actual knowledge that is to be exchanged between agents. KQML is independent of knowledge representing language and ontology used to describe the content of the message. The content can be represented in any language including KIF and XML. The communication layer contains information for communicating messages between agents and tracking their conversations. These parameters include unique identifiers associated with the sender and receiver and each conversation thread. An example of a KQML message is shown below.

```
(tell
  :sender          agent02        :receiver     agent01
  :in-reply-to  message11   :language    KIF
  :ontology          map              :content      position (21,60)
)
```

This message is sent from *agent02* to *agent01* in reply to a previous message with the ID of *message11*. The content of the message is to give information about the position of target along the x and y axis. The performative in this example is *'tell'*, which is an informative performative.

Several versions of KQML have been developed consisting of a different collection of performatives [14]. Some examples of the KQML performatives include:

- Basic queries: *ask-if*, *ask-about*, *reply*.
- Basic informative: *tell*, *deny*.
- Database queries: *insert*, *delete*, *update*.
- Network queries: *broadcast*, *resister*, *transport-addresses*.

Facilitator Knowledge Query Manipulation Language (KQML) provides an architecture for agents to communicate through a communication facilitator or matchmaker [15]. The facilitator agent plays an important role in Multi-Agent System as it offers various useful communication and network services that allow agents to interact. Agents can communicate and register themselves with the facilitator using the 'advertise' performative. The facilitator can identify agents via their name, address, and capabilities. It can recommend agents to each other based upon their requests and based on the particular service or information they provide. In other words, the facilitator provides a centralized meeting place for agents. It may also be used to transfer information between agents. KQML has been used extensively by the research community and in many Multi-Agent System applications. A number of implementations have been developed, which include JACKAL [16], InfoSleuth [17], KAoS [18], and JATLite [19]. JACKAL is a Java implementation of KQML developed at the University of Maryland.

KQML has been criticized for a number of reasons. It lacks formal and adequate semantics. There are different interpretations of the meaning of its performatives [14]. The transport mechanism of messages in KQML is not defined accurately, which makes it difficult for different KQML-speaking agents to interoperate. It is missing the 'commissive' performatives that are important if agents need to make commitments to and coordinate their actions with each other [7]. There are no organized efforts to maintain and further develop KQML. In fact, KQML efforts have been subsumed by activities of FIPA to extend and modify it for use as basic ACL in the FIPA standards.

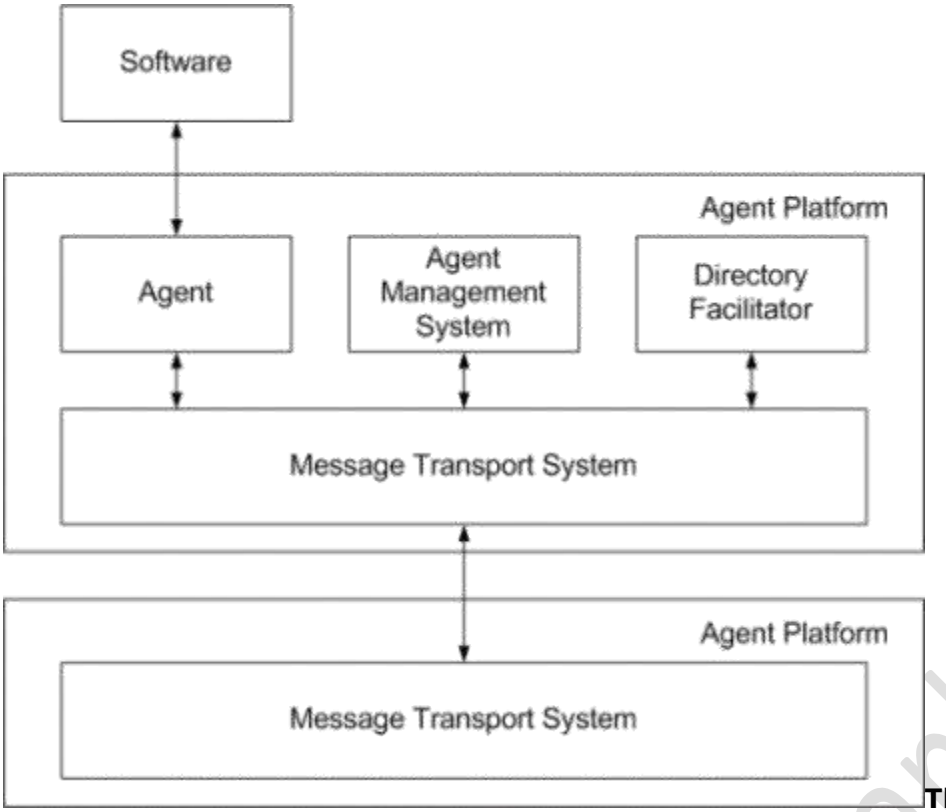
Foundation of Intelligent Physical Agents

The Foundation of Intelligent Physical Agents (FIPA) institution develops standards for agent systems [20]. It has played an important role in defining agent communication standards. Its aim is to promote agent-based technology and produce specifications on Agent Communication Languages [1]. The first set of FIPA specifications were released in 1997 [21]. The specifications required CORBA Internet Inter-ORB Protocol (IIOP) as the communication protocol.

CORBA enables the communication between software components and procedures that are written in different languages on different computers [10]. With the emergence of alternative communication protocols including HTML and Java RMI, FIPA developed a new set of technology-independent specifications.

FIPA Agent Communication Language is the communication language for agents developed by FIPA. It is based on the Speech Act theory and represents messages in a structure in an Envelope format similar to KQML. Important elements of a FIPA ACL message include performative, sender, receiver, and content. Other elements include language, ontology, protocol, and conversation identification. The collection of performatives that FIPA ACL provides is different from KQML. The core performatives are *'inform'* and *'request'*. A set of performatives have been defined that allows agents to perform various tasks including sharing knowledge, participating in various conversations, and negotiating with other agents. Some examples of performatives in FIPA Agent Communication Language include [14]:

- Passing on information: *confirm*, *disconfirm*, *inform*, *inform-if*.
- Requesting information: *cancel*, *query-if*, *subscribe.*
- *Negotiation: accept-proposal, propose, reject-proposal.*
- *Performing actions: agree, cancel, propagate, refuse. request.*
- *Error handling: failure, not understood*.



The FIPA agent platform components [22].

The FIPA Agent Platform model provides Agent Management System (AMS), Directory Facilitator (DF), and a Message Transport System (MTS) [22], as shown in Figure above. AMS controls the life-cycle of agents, which includes their creation, registration, communication, migration, and retirement. Directory Facilitators are important components of the FIPA Agent Platform model that provide Yellow Pages Lookup Services to other agents. Directory Facilitator registers agents based on the services they offer. It also recommends relevant agents to other agents that look for particular services. Message Transport System provides internal communication methods between agents and external messaging with other FIPA-compliant Agent Platforms.

Unlike KQML, FIPA provides comprehensive semantics to the language. The semantic language of FIPA Agent Communication Language is a LISP-like language called Semantic Language (SL), which allows encoding of concepts and actions that agents perform. FIPA specification is based on the BDI (Beliefs, Desires, and Intentions) agent model [23], which considers agents to have mental states. The BDI model originated from a philosophical theory of practical reasoning [24]. Beliefs are the understanding of an agent about the environment and other agents [25]. Desires are the goals of an agent related to what it wants to achieve. Intentions are its plans for taking action to achieve the goal(s). Despite the popularity of FIPA in the early 2000s, it did not succeed in gaining commercial support [4]. Very few software tools have been developed for agents that comply with FIPA specifications. The most popular of them include *JACK* and *JADE*.

Simple Object Access Protocol

Simple Object Access Protocol (SOAP) is a communication protocol independent of the platform, operating system, and programming language [2]. It is a simple and extensible communication protocol that is based on XML. In the context of web services, SOAP is considered as "the standard message protocol for exchanging XML data over the Internet" [4]. It allows complex interactions between web services through request/response exchanges.

SOAP has been used in this research to provide a universal translator approach for communication and tasking. It provides an increased level of interoperability for Multi-Agent Systems.

A SOAP message is essentially an XML document that consists of two main elements, *Envelope* and *Body*, and the optional *Header* and *Fault* elements [26]:

- An *Envelope* is the root element of every SOAP document that wraps the message. It contains attributes that identify the XML document is a SOAP message and specifies its encoding style.
- The *Body* element includes the content of information intended for the recipient. The content represents knowledge using XML by defining meaningful tags and wrapping them around the relevant information. The content contains XML elements specific to the service.
- The *Header* element may be used to state application-specific information about the message. For example, it may include information about the message's sender and recipient, or represent a performative describing the intention of the sender of the message. There also are some

predefined attributes for the Header element that defines how the message should be processed by the recipient.
- The *Fault* element contains errors and status information. It is used to identify faults and error messages and what the causes are.

Below is a sample SOAP request message that invokes a web service to get the contact information of a user given their name.

```
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Header>  ...  </soap:Header>

  <soap:Body xmlns:m="http://www.example.org/profiles">

     <m:GetPhoneNumber>
        <m:Name>Mr Smith</m:Name>
     </m:GetPhoneNumber>

     <soap:Fault>  ...  </soap:Fault>

  </soap:Body>

</soap:Envelope>
```

The content of the message contains XML elements specific to this application, which include *'m:GetPhoneNumber'* and *'m:Name'*. These are application specific elements defined in the namespace of *'http://www.example.org/profiles'*. This example requests the phone number of a user named Mr Smith.

The Header and Fault elements are optional and are shown in this example only to show their location in the XML document. If a header element is used in a SOAP message, it must be the first child element of the Envelope [27]. The Fault element can only appear once and only as a child element of the Body. The response message of the SOAP message above may look like this.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPhoneNumberResponse xmlns:m="http://www.example.org/profiles">
    <m:Phone>83232923</m:Phone>
  </m:GetPhoneNumberResponse>
</soap:Body>

</soap:Envelope>
```

- **KIF**: Knowledge Interchange Format
- **XML**: Extensible Mark-up Language
- **KQML**: Knowledge Query Manipulation Language
- **FIPA**: Foundation of Intelligent Physical Agents
- **SOAP**: Simple Object Access Protocol

Applications of intelligent agents

Intelligent agents in artificial intelligence have been applied in many real-life situations.

Information search, retrieval, and navigation

Intelligent agents enhance the access and navigation of information. This is achieved through the search of information using search engines. The internet consists of many data objects that may take users a lot of time to search for a specific data object. Intelligent agents perform this task on behalf of users within a short time.

Repetitive office activities

Some companies have automated certain administrative tasks to reduce operating costs. Some of the functional areas that have been automated include customer support and sales. Intelligent agents have also been used to enhance office productivity.

Medical diagnosis

Intelligent agents have also been applied in healthcare services to improve the health of patients. In this case, the patient is considered as the environment. The computer keyboard is used as the sensor that receives data on the symptoms of the patient. The intelligent agent uses this information to decide the best course of action. Medical care is given through actuators such as tests and treatments.

Vacuum cleaning

AI agents are also used to enhance efficiency and cleanness in vacuum cleaning. In this case, the environment can be a room, table, or carpet. Some of the sensors employed in vacuum cleaning include cameras, bump sensors, and dirt detection sensors. Action is initiated by actuators such as brushes, wheels, and vacuum extractors.

Autonomous driving

Intelligent agents enhance the operation of self-driving cars. In autonomous driving, various sensors are employed to collect information from the environment. These include cameras, GPS, and radar. In this application, the environment can be pedestrians, other vehicles, roads, or road signs. Various actuators are used to initiate actions. For example, brakes are used to bring the car to a stop.

CHAPTER IX : ADVANCED KNOWLEDGE REPRESENTATION TECHNIQUES

Topics covered: Conceptual dependency theory, script structures, CYC theory, script structure, case grammars, semantic web.

Conceptual Dependency (CD)

Conceptual Dependency originally developed to represent knowledge acquired from natural language input.

The goals of this theory are:

- To help in the drawing of inference from sentences.
- To be independent of the words used in the original input.
- That is to say: *For any 2 (or more) sentences that are identical in meaning there should be only one representation of that meaning.*

It has been used by many programs that portend to understand English (*MARGIE, SAM, PAM*). CD developed by Schank *et al.* as were the previous examples.

CD provides:

- a structure into which nodes representing information can be placed
- a specific set of primitives
- at a given level of granularity.

Sentences are represented as a series of diagrams depicting actions using both abstract and real physical situations.

- The agent and the objects are represented
- The actions are built up from a set of primitive acts which can be modified by tense.

Examples of Primitive Acts are:

ATRANS
-- Transfer of an abstract relationship. *e.g. give*.
PTRANS
-- Transfer of the physical location of an object. *e.g. go*.
PROPEL
-- Application of a physical force to an object. *e.g. push*.
MTRANS
-- Transfer of mental information. *e.g. tell*.
MBUILD
-- Construct new information from old. *e.g. decide*.
SPEAK
-- Utter a sound. *e.g. say*.
ATTEND
-- Focus a sense on a stimulus. *e.g. listen, watch*.
MOVE
-- Movement of a body part by owner. *e.g. punch, kick*.
GRASP
-- Actor grasping an object. *e.g. clutch*.
INGEST
-- Actor ingesting an object. *e.g. eat*.
EXPEL
-- Actor getting rid of an object from body. *e.g. ????*.

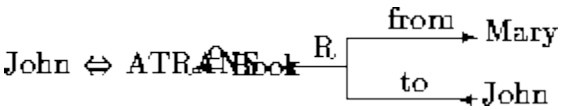
Six primitive conceptual categories provide *building blocks* which are the set of allowable dependencies in the concepts in a sentence:

PP
-- Real world objects.
ACT
-- Real world actions.
PA
-- Attributes of objects.
AA
-- Attributes of actions.
T
-- Times.
LOC
-- Locations.

How do we connect these things together?

Consider the example:

John gives Mary a book

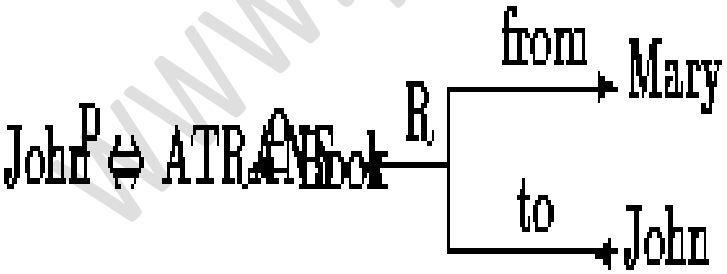


- Arrows indicate the direction of dependency. Letters above indicate certain relationships:

o
-- object.
R
-- recipient-donor.
I
-- instrument *e.g. eat with a spoon*.
D
-- destination *e.g. going home*.

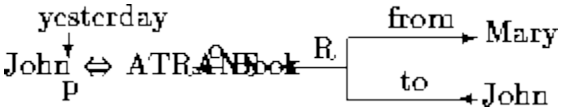
- Double arrows (⇔) indicate *two-way* links between the actor (PP) and action (ACT).
- The actions are built from the set of primitive acts (see above).
  - These can be modified by *tense etc.*

The use of tense and mood in describing events is extremely important and schank introduced the following modifiers:
p
-- past
f
-- future
t
-- transition
$\mathbf{t_s}$
-- start transition
$\mathbf{t_f}$
-- finished transition
k
-- continuing
?
-- interrogative
/
-- negative
delta
-- timeless
c
-- conditional
the absence of any modifier implies the *present tense*.
So the *past tense* of the above example:
John gave Mary a book becomes:



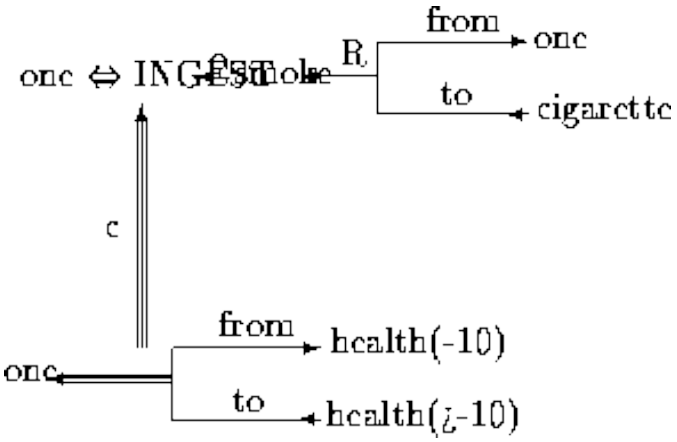
The ⇔ has an object (actor), PP and action, ACT. *I.e.* PP ⇔ ACT. The triplearrow (⟺) is also a two link but between an object, PP, and its attribute, PA. *I.e.* PP ⟺ PA.
It represents *isa* type dependencies. *E.g*
Dave ⟺ lecturerDave is a lecturer.
Primitive states are used to describe many state descriptions such as height, health, mental state, physical state.
There are many more physical states than primitive actions. They use a numeric scale.
E.g. John ⟺ height(+10) *John is the tallest* John ⟺ height(< average) *John is short* Frank Zappa ⟺ health(-10) *Frank Zappa is dead* Dave ⟺ mental_state(-10) *Dave is sad* Vase ⟺ physical_state(-10) *The vase is broken*

You can also specify things like the time of occurrence in the relationship.
For Example: *John gave Mary the book yesterday*



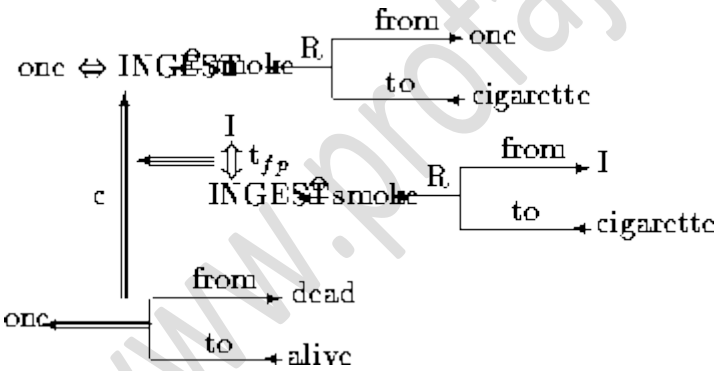
Now let us consider a more complex sentence: *Since smoking can kill you, I stopped* Lets look at how we represent the inference that *smoking can kill*:

- Use the notion of *one* to apply the knowledge to.
- Use the *primitive act* of INGESTing smoke from a cigarette to one.
- Killing is a transition from being alive to dead. We use *triple arrow*s to indicate a transition from one state to another.
- Have a conditional, *c* causality link. The *triple arrow* indicates dependency of one concept on another.



To add the fact that *I stopped smoking*

- Use similar rules to imply that I smoke cigarettes.
- The qualification t_{fp} attached to this dependency indicates that the instance INGESTing smoke has stopped.



Advantages of CD:

- Using these primitives involves fewer inference rules.
- Many inference rules are already represented in CD structure.
- The holes in the initial structure help to focus on the points still to be established.

Disadvantages of CD:

- Knowledge must be decomposed into fairly low level primitives.
- Impossible or difficult to find correct set of primitives.
- A lot of inference may still be required.
- Representations can be complex even for relatively simple actions. Consider:

Dave bet Frank five pounds that Wales would win the Rugby World Cup.
Complex representations require a lot of storage

Applications of CD:
MARGIE
(*Meaning Analysis, Response Generation and Inference on English*) -- model natural language understanding.
SAM
(*Script Applier Mechanism*) -- Scripts to understand stories. See next section.
PAM
(*Plan Applier Mechanism*) -- Scripts to understand stories.
Schank *et al.* developed all of the above.

A number of authors in AI have addressed the question of the 'concept'-based organisation of knowledge and we use two examples to illustrate this.
•We consider a verb-oriented organisation of knowledge proposed by Schank: Conceptual Dependency Grammar
Conceptual dependency (or CD) is a theory of how to represent the meaning of natural language sentences in a way that:
•First, facilitates for drawing inferences from the sentences.
•Second, it has been argued that the representation (CD) is independent of the language in which the sentences were originally stated

•Schank's (1975) Conceptual Dependency Theory was developed as a part of a natural language comprehension project.
•Schank's claim was that sentences can be translated into basic concepts expressed as a small set of semantic primitives.
•Conceptual dependency allows these primitives, which signify meanings, to be combined to represent more complex meanings.
•Schank calls the meaning propositions underlying language "conceptualisations".

Roger Schank's project is the 'representation of meaning in an unambiguous language-free manner' (1973). 'Any two utterances that can be said to mean the same thing, whether they are in the same or different languages, should be characterised in only one way by the conceptual structure' (1973) (CG) Three elemental kinds of concepts: a nominal and an action together with their modifiers

Three elemental kinds of concepts:

- an abstract or concrete object that invokes an image;
- *cars* are concrete objects;
- *gravity* is an abstract concept;
- a nominal produces a picture (PP)

2. Concept —can be→ an action

- what a nominal does?
- something an animate nominal does to an object;
- there are primitive ACTions and derived ACTions;

3. Concept —can be→ a modifier

- a modifier modifies a nominal or an action;
- a modifier specifies an action or a nominal;

3a Concept —can be→ a picture modifier or aider (PA)

- a blue car
  PA

3b Concept —can be→ an action modifier or aider (AA)

- he quickly run
  AA

CD theorists argue that 'the CD representation of a sentence is built not out of primitives corresponding to the words used in the sentence, but rather out of conceptual primitives that can be combined to form the meanings of words in any particular language'

CD Building Blocks
Primitive Acts
Primitive conceptual categories
Conceptual Tenses
Diagrammatic conventions

Conceptual Categories Three elemental kinds of concepts – conceptual categories (**PP, ACT, PA and AA**) – relate to each other in specified ways. These relations are called dependencies by Schank.

Conceptual Categories In a dependency relation, one partner or item is dependent and the other dominant or governing; a governor↔dependent is a partially ordered relationship
•A **dependent** must have a governor and is understood in terms of the governor
•A **governor may or may not have dependent(s)** and has an independent existence
•A **governor can be a dependent PP and ACT are inherently governing categories**, PA and AA are inherently dependent.

Conceptual Categories For a conceptualisation to exist, there must be at least two governors:

Sally stroked her fat cat

'Sally' name of an object (nominal ⇒ PP)
'stroked' name of an action (ACT)

PP:	**Sally, cat, her [Sally]**
ACT:	**to stroke**
PA:	**fat**
Governors:	**Sally, stroke, cat**
Dependent:	**PP (cat) on ACT (stroke)**
	PA (fat) on PP (cat)
	PP (cat) on PP (her[Sally])

Conceptual Categories

Sally stroked her fat cat

1. **Sally and stroking are necessary for conceptualisation: there is a two-way dependency between each other:**

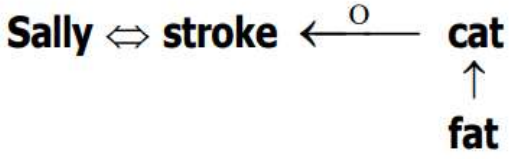
   Sally ⇔ stroke

2. **Sally's cat cannot be conceptualised without the ACT *stroke* ⇒ it has an objective dependency on *stroke***

   Sally ⇔ stroke $\xleftarrow{O}$ cat.

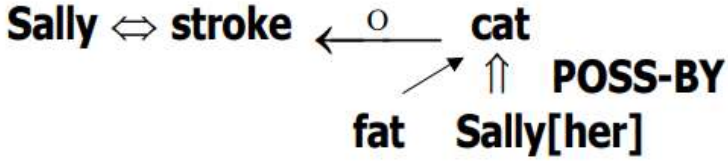
Sally stroked her fat cat

3. The concept 'cat' is the governor for the modifier 'fat':



Sally stroked her fat cat

4. The concept PP(cat) is also governed by the concept PP(Sally) through a *prepositional dependency*:



A Conceptual Dependency Network

Conceptual Cases

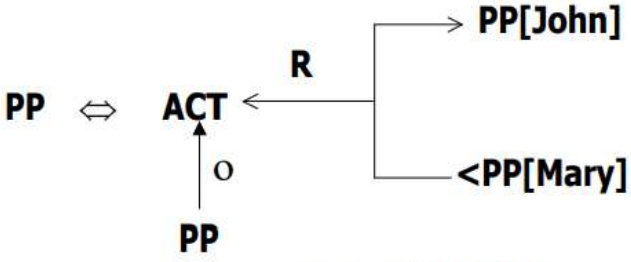
Dependents that are required by an ACT are called *Conceptual Cases*: Consider the following sentences:

Objective Case (O)

1. John took the book

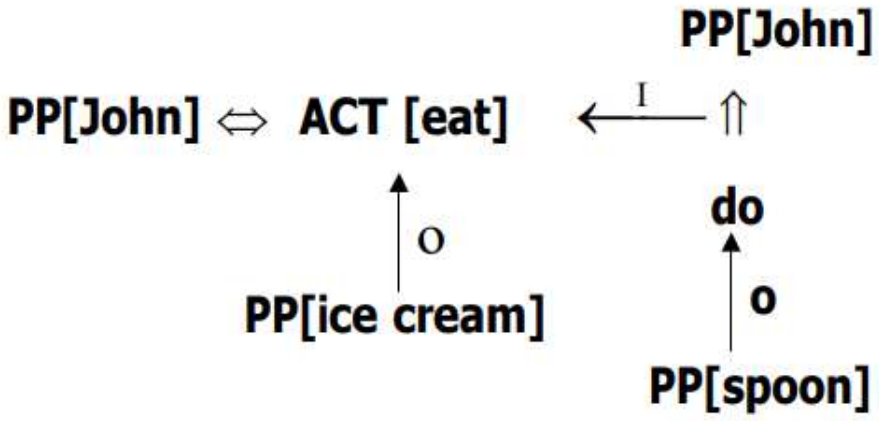
Recipient Case (R)

2. John took the book from Mary



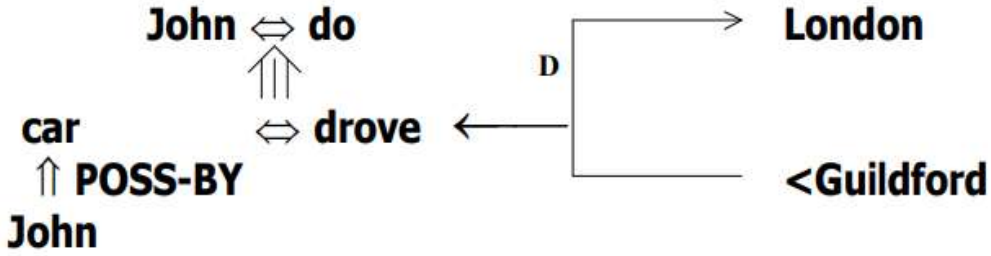
Instrumental Case (I)

3. John ate the ice cream with a spoon



Directive Case Relation (D)

4. John drove his car to London from Guildford



(⇑) indicates causality

Prepositional Dependency

Consider the following sentences:

Possession

- **This is Sally's cat ⇒ one PP (Sally) provides specific information about another PP (cat) -**

Cat
⇑ POSS-BY
Sally

Location

- **Sally is in London -**

London
⇑ LOC
Sally

Containment

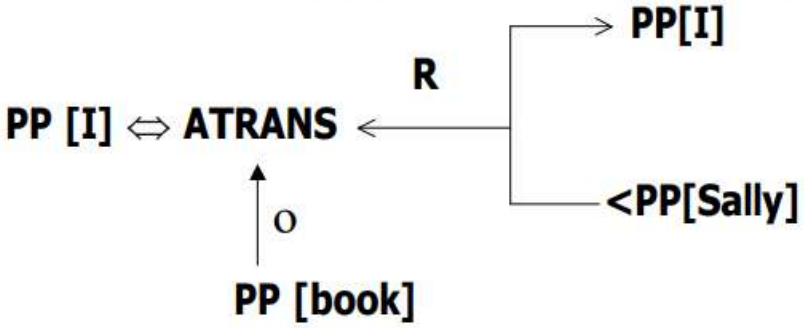
- **The glass contains water -**

Water
⇑ CONT
Glass

Underlying ACTs

Giving and Taking OR just TRANSferring

I took a book from Sally



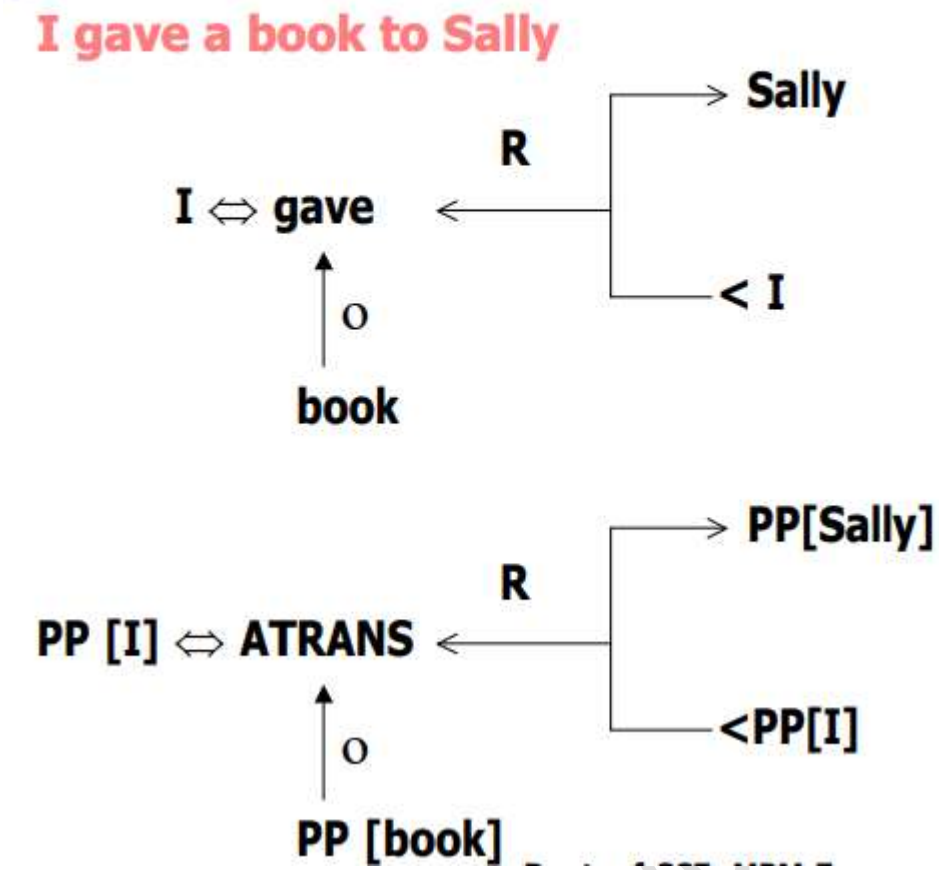
Sentence = 'I took a book from Sally'

CD Building Block	Elaboration
Primitive Acts	ATRANS indicates *transfer* (of possession)
Primitive conceptual Categories	Objects (Picture Producers: PP): Sally, I, book
Conceptual Tenses	o indicates *object case* relation; R indicates *recipient case* relation
Diagrammatic Conventions	Arrows indicate the *direction* of dependency Double arrow indicates *two way* link between *actor* and *action*

Underlying ACTions

Primitive Act	Elaboration
ATRANS	Transfer of an abstract relationship such as possession ownership or control (give)
PTRANS	Transfer of the physical location of an object (go)
PROPEL	Application of a physical force to an object (push)
MOVE	Movement of a body part of an animal by that animal (kick)
GRASP	Grasping of an object by an actor (grasp)
INGEST	Taking in of an object by an animal to the inside of that animal (eat)
EXPEL	Expulsion of an object from the object of an animal into the physical world (cry)
MTRANS	Transfer of mental information between animals or within an animal (tell)
MBUILD	Construction by an animal of new information of old information (decide)
CONC	Conceptualise or think about an idea (think)
SPEAK	Actions of producing sounds (say)
ATTEND	Action of attending or focusing a sense organ towards a stimulus (listen)

Underlying ACTs



Underlying ACTs – The TRANSfer ACT

- Transfer of an abstract relationship: possession of property ⇒ give, take; Abstract TRANSfer (ATRANS)
- Transfer of the physical location of an object ⇒ go, come; Physical TRANSfer (PTRANS)
- Transfer of mental information ⇒ ask, tell; Mental TRANSfer (MTRANS)

Underlying ACTs – STATICS and DYNAMICS

- PROPELling an object by applying physical force ⇒ push, pull; PROPEL
- MOVEment of a body part by its owner ⇒ kick; MOVE
- GRASPing of an object by an actor ⇒ catching, clutching; GRASP

Underlying ACTs – Sustaining life; processing things

- INGESTing of an object by an animal ⇒ eat, drink; INGEST
- EXPELling of something from a body of an animal ⇒ crying [tears]

Underlying ACTs – Stimulus and Response

- Mentally BUILDing new information out of old ⇒ deciding, inferring; MBUILD
- SPEAKing by producing sounds ⇒ talking; SPEAK
- ATTENDing to a stimulus by focusing a sense organ towards it ⇒ seeing, hearing; ATTEND

Conceptual Tenses

Any conceptualisation can be modified as a whole by a *conceptual tense*.

John took the book – John ⇔ took – can be denoted by looking at the lemma *take* (from which the past tense *took* was derived):

$$\text{John} \overset{p(\text{ast})}{\Leftrightarrow} \text{take}$$

Conceptual Tenses

Symbol	Elaboration
p	Past
f	Future
t	Transition
t_s	Start Transition
t_f	Finished Transition
k	Continuing
?	Interrogative
/	Negative
nil	Present
Delta	Timeless
c	Conditional

•John will be taking the book can be described as:

John ⇔ taking *or* John ⇔ take$^{f(uture)}$

•John is taking the book can be described as:

John ⇔ taking *or* John ⇔ take^{k}

Semantic Nets and Conceptual Dependency Representation:
A comparison
• Semantic Nets only provide a structure into which nodes representing information can be placed
• Conceptual Dependency representation, on the other hand, provides both a structure and a specific set of primitives out of which representations of particular pieces of information can be constructed

Advantages of using Conceptual Dependency Grammar for representing knowledge and reasoning with a CD knowledge base

1. The organisation of knowledge in terms of the primitives (or 'primitive acts') leads to a fewer inference rules.

2. Many inferences are already contained in the representation itself.

3. The initial structure that is built to represent the information contained in one sentence will have holes in it that have to be filled in: holes(e.g. book, john, etc.) which will serve as attention focusers for subsequent sentences.

Disadvantages of using Conceptual Dependency Grammar for representing knowledge and reasoning with a CD knowledge base

1. CD requires all knowledge to be broken down into 11 primitives: sometimes inefficient and sometimes impossible

2. CD is essentially a theory of the representation of events: though it is possible to have an event centred view of knowledge but not a practical proposition for storing and retrieving knowledge

3. May be difficult or impossible to design a program that will reduce sentences to canonical form. (Provably not possible for monoids, which are simpler than natural language).

4. Computationally expensive to reduce all sentences to the primitives.

Script Structures:

A script is a structured representation describing a stereotyped sequence of events in a particular context.

Scripts are used in natural language understanding systems to organize a knowledge base in terms of the situations that the system should understand. Scripts use a frame-like structure to represent the commonly occurring experience like going to the movies eating in a restaurant, shopping in a supermarket, or visiting an ophthalmologist.

Thus, a script is a structure that prescribes a set of circumstances that could be expected to follow on from one another.

Scripts are beneficial because:

- Events tend to occur in known runs or patterns.
- A casual relationship between events exist.
- An entry condition exists which allows an event to take place.
- Prerequisites exist upon events taking place.

Components of a script

The components of a script include:

- **Entry condition:** These are basic condition which must be fulfilled before events in the script can occur.
- **Results:** Condition that will be true after events in script occurred.
- **Props:** Slots representing objects involved in events
- **Roles:** These are the actions that the individual participants perform.
- **Track:** Variations on the script. Different tracks may share components of the same scripts.
- **Scenes:** The sequence of events that occur.

Describing a script, special symbols of actions are used. These are:

Symbol	Meaning	Example
ATRANS	transfer a relationship	give
PTRANS	transfer physical location of an object	go
PROPEL	apply physical force to an object	push
MOVE	move body part by owner	kick
GRASP	grab an object by an actor	hold
INGEST	taking an object by an animal eat	drink
EXPEL	expel from animal's body	cry
MTRANS	transfer mental information	tell
MBUILD	mentally make new information	decide
CONC	conceptualize or think about an idea	think
SPEAK	produce sound	say
ATTEND	focus sense organ	listen

Example:-Script for going to the bank to withdraw money.
SCRIPT : Withdraw money
TRACK : Bank
PROPS : Money
Counter
Form
Token
Roles : P= Customer
E= Employee
C= Cashier
Entry conditions: P has no or less money.
The bank is open.
Results : P has more money.
Scene 1: Entering
P PTRANS P into the Bank
P ATTEND eyes to E
P MOVE P to E
Scene 2: Filling form
P MTRANS signal to E
E ATRANS form to P
P PROPEL form for writing
P ATRANS form to P
E ATRANS form to P
Scene 3: Withdrawing money
P ATTEND eyes to counter
P PTRANS P to queue at the counter
P PTRANS token to C
C ATRANS money to P
Scene 4: Exiting the bank
P PTRANS P to out of bank

Advantages of Scripts

- Ability to predict events.
- A single coherent interpretation maybe builds up from a collection of observations.

Disadvantages of Scripts

- Less general than frames.
- May not be suitable to represent all kinds of knowledge

CYC
What is CYC?
• An ambitious attempt to form a very large knowledge base aimed at capturing commonsense reasoning.
• Initial goals to capture knowledge from a hundred randomly selected.
• Both Implicit and Explicit knowledge encoded. Example: Suppose we read that Wellington learned of Napoleon's death Then we (humans) can conclude Napoleon never new that Wellington had died. How do we do this?

We require special implicit knowledge or commonsense such as:
• We only die once.
• You stay dead.
• You cannot learn of anything when dead.
• Time cannot go backwards. Why build large knowledge bases:

Brittleness -- Specialised knowledge bases are brittle. Hard to encode new situations and non-graceful degradation in performance.
Commonsense based knowledge bases should have a firmer foundation.
Form and Content -- Knowledge representation may not be suitable for AI.
Commonsense strategies could point out where difficulties in content may affect the form. Shared **Knowledge --** Should allow greater communication among systems with common bases and assumptions.

How is CYC coded?

- By hand.
- Special CYCL language:
  - o LISP like.
  - o Frame based o Multiple inheritance
  - o Slots are fully fledged objects.
  - o Generalised inheritance -- any link not just isa and instance.

CASE GRAMMAR

• Case grammar is a system of linguistic analysis, focusing on the link between the valence, or number of subjects, objects, etc., of a verb and the grammatical context it requires. The system was created by the American linguist Charles J. Fillmore in (1968), in the context of Transformational Grammar. This theory analyzes the surface syntactic structure of sentences by studying the combination of deep cases (i.e. semantic roles) -- Agent, Object, Benefactor, Location or Instrument—which are required by a specific verb.

For instance, the verb "give" in English requires an **Agent (A)** and **Object (O),** and a **Beneficiary (B)**; e.g**. "Jones (A) gave money (O) to the school (B).**

• According to Fillmore, each verb selects a certain number of deep cases which form its case frame. Thus, a case frame describes important aspects of semantic valency, of verbs, adjectives and nouns. Case frames are subject to certain constraints, such as that a deep case can occur only once per sentence. Some of the cases are obligatory and others are optional. Obligatory cases may not be deleted, at the risk of producing ungrammatical sentences. For example, Mary gave the apples is ungrammatical in this sense.

• A fundamental hypothesis of case grammar is that grammatical functions, such as subject or object, are determined by the deep, semantic valence of the verb, which finds its syntactic correlate in such grammatical categories as Subject and Object, and in grammatical cases such as Nominative, Accusative, etc. Fillmore (1968) puts forwards the following hierarchy for a universal subject selection rule:

• Agent < Instrumental < Objective

• That means that if the case frame of a verb contains an agent, this one is realized as the subject of an active sentence; otherwise, the deep case following the agent in the hierarchy (i.e. Instrumental) is promoted to subject.

• The influence of case grammar on contemporary linguistics has been significant, to the extent that numerous linguistic theories incorporate deep roles in one or other form, such as the so-called Thematic structure in Government and Binding theory. It has also inspired the development of frame-based representations in AI research

Fillmore Grammar is also called as case grammar. Case grammar provides a different approach to the problem of how syntactic and semantic interpolation can be combined. Grammar rules are written to describe syntactic rather than semantic regularities. But the structures, and rules that are produced correspond to semantic relations rather than to strictly syntactic ones.

Case grammar describes the relationships between verbs and their arguments parsing using case grammar is usually expectation-driven. once the verb of the sentence is located, it can be used to predict the noun phrases that will occur and to determine the relationship of those phrases to the rest of the sentences.

A case grammar describes the correct set of deep cases.

Characteristics of Fillmore Case Grammar:

i. Agent: Investigation of the action.
ii. Instrument: Cause of the event.
iii. Dative: Entity affected by the action.
iv. Factitive: Object.
v. Locative: Place of the event.
vi. Source: Place from which something moves.

SEMATIC WEB:

The Semantic Web is an extension of the World Wide Web through standards set by the World Wide Web Consortium (W3C).

The goal of the Semantic Web is to make Internet data machine-readable. To enable the encoding of semantics with the data, technologies such as Resource Description Framework (RDF) and Web Ontology Language (OWL) are used.

These technologies are used to formally represent metadata. For example, ontology can describe concepts, relationships between entities, and categories of things.
These embedded semantics offer significant advantages such as reasoning over data and operating with heterogeneous data sources.
These standards promote common data formats and exchange protocols on the Web, fundamentally the RDF. According to the W3C, "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries."
The Semantic Web is therefore regarded as an integrator across different content and information applications and systems.
The term was coined by Tim Berners-Lee for a web of data (or data web) that can be processed by machines—that is, one in which much of the meaning is machine-readable. While its critics have questioned its feasibility, proponents argue that applications in library and information science, industry, biology and human sciences research have already proven the validity of the original concept.
In addition to the classic "Web of documents" W3C is helping to build a technology stack to support a "Web of data," the sort of data you find in databases.
The ultimate goal of the Web of data is to enable computers to do more useful work and to develop systems that can support trusted interactions over the network.
The term "Semantic Web" refers to W3C's vision of the Web of linked data.
Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data. Linked data are empowered by technologies such as RDF, SPARQL, OWL, and SKOS.
Semantic Web is an extension to the World Wide Web. The purpose of the semantic web is to provide structure to the web and data in general. It emphasizes on representing a web of data instead of web of documents. It allows computers to intelligently search, combine and process the web content based on the meaning that the content has. Three main models of the semantic web are:

1. Building models
2. Computing with Knowledge
3. Exchanging Information

- **Building Models:**
  Model is a simplified version or description of certain aspects of the real-time entities. Model gathers information which is useful for the understanding of the particular domain.
- **Computing Knowledge:**
  Conclusions can be obtained from the knowledge present.
  Example: If two sentences are given as *'John is the son of Harry'* and another sentence given is- *'Hary's father is Joey'*, then the knowledge that can be computed from it is – *'John is the grandson of Joey'*
  Similarly, another example useful in the understanding of computing knowledge is-
  'All A is B' and *'All B is C'*, then the conclusion that can be drawn from it is – *'All A are C'* respectively.
- **Exchanging Information:**
  It is an important aspect. Various communication protocols have been implemented for the exchange of information like the TCP/IP, HTML, WWW. Web Services have also been used for the exchange of the data.

The technologies associated with the semantic web are:

- RDF (Resource Description Framework)
- OWL (Web Ontology Language)
- DL (Description Language)

The query language used is:

- SPARQL (SPARQL Protocol and RDF query language).
- SHACL (Shape Constraint Language). SHACL is used for validating the RDF graphs against a set of conditions.

Criteria	Semantic Web	AI
Definition	A vision of a web where data is interconnected and machine-readable.	A field of computer science that aims to create intelligent machines capable of performing tasks that typically require human intelligence.
Goal	To enable machines to understand	To create intelligent machines capable of

	and interpret data on the web.	performing tasks that require human intelligence, such as reasoning, problem-solving, perception, and natural language processing.
Focus	Interoperability, standardization, and data integration.	Intelligent decision-making, learning, and perception.
Approach	Adding metadata and annotations to web resources to make them machine-understandable.	Developing algorithms and models that can learn from data, reason, and make decisions.
Technologies	Resource Description Framework (RDF), Web Ontology Language (OWL), and Semantic Web Rule Language (SWRL).	Machine learning, deep learning, natural language processing (NLP), computer vision, and robotics.
Benefits	Better integration and discovery of data, improved search results, and increased automation.	Enhanced decision-making, improved customer experience, and increased efficiency and automation.
Challenges	The need for a shared understanding of concepts, the high cost of creating and maintaining ontologies, and the limited adoption of Semantic Web technologies.	The risk of bias and errors in decision-making, the challenge of interpreting and explaining results, and the ethical implications of AI-powered systems.
Examples	Schema.org, Linked Open Data Cloud, and OpenCyc etc.	Siri, Alexa, chatbots, autonomous vehicles, and predictive analytics etc.

CHAPTER X: NATURAL LANGUAGE PROCESSING

Topic covered:

Sentence Analysis phases, grammars and parsers, types of parsers, semantic analysis, universal networking language, dictionary

---

Natural Language Processing (NLP) a subset technique of Artificial Intelligence which is used to narrow the communication gap between the Computer and Human.

It is originated from the idea of Machine Translation (MT) which came to existence during the 1950.

The primary aim was to translate one human language to another human language, for example, Russian language to English language using the brain of the Computers but after that, the thought of conversion of human language to computer language and vice-versa emerged, so that communication with the machine became easy.

2 Natural Language Processing NLP is a process where input provided in a human language and converts this input into a useful form of representation.

3 The field of NLP is primarily concerned with getting computers to perform interesting and useful tasks with human languages.

4 The field of NLP is secondarily concerned with helping us come to a better understanding of human language. As stated above the idea had emerged from the need for Machine Translation in the 1950s.

5 Then the original language was English and Russian. But the use of other words such as Chinese also came into existence in the initial period of the 1960s. In the 1960s, the NLP got a new life when the idea and need of Artificial Intelligence emerged. In 1978 LUNAR is developed by W.A woods; it could analyze, compare and evaluate the chemical data on a lunar rock and soil composition that was accumulating as a result of Apollo moon missions and can answer the related question. In the 1980s the area of computational grammar became a very active field of research which was linked with the science of reasoning for meaning and considering the user's beliefs and intentions. In the period of 1990s, the pace of growth of NLP increased. Grammars, tools and Practical resources related to NLP became available with the parsers. Probabilistic and data-driven models had become quite standard by then. In 2000, Engineers had a large amount of spoken and textual data available for creating systems. Today, large amount of work is being done in the field of NLP using Machine Learning or Deep Neural Networks in general, where we are able to create state-of-the-art models in text classification, Question and Answer generation, Sentiment Classification, etc.

There are general five steps in natural language processing

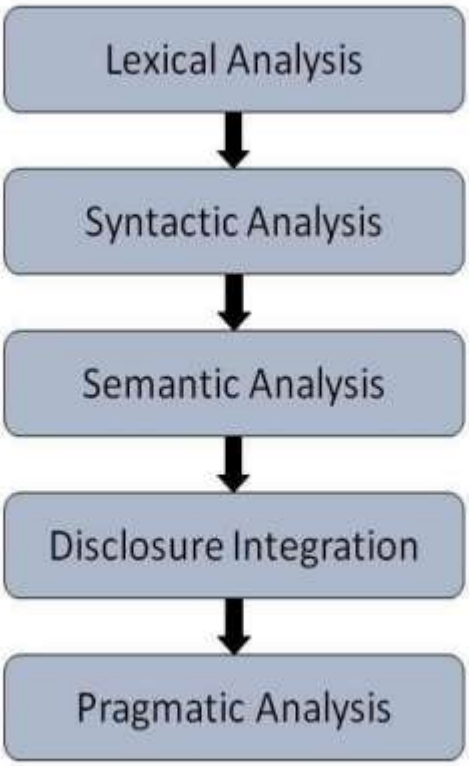


Figure 1.6.1 General five steps of NLP

1) **Lexical Analysis**: It is the first stage in NLP. It is also known as morphological analysis. It consists of identifying and analyzing the structure of words. Lexicon of a language means the collection of phrases and words in a language. Lexical analysis is dividing the whole chunk of text into words, sentences, and paragraphs

2) **Syntactic Analysis**: Syntactic analysis consists of analysis of words in the sentence for grammar and ordering words in a way that shows the relationship among the words. For example the sentence such as "The school goes to boy" is rejected by English syntactic analyzer.

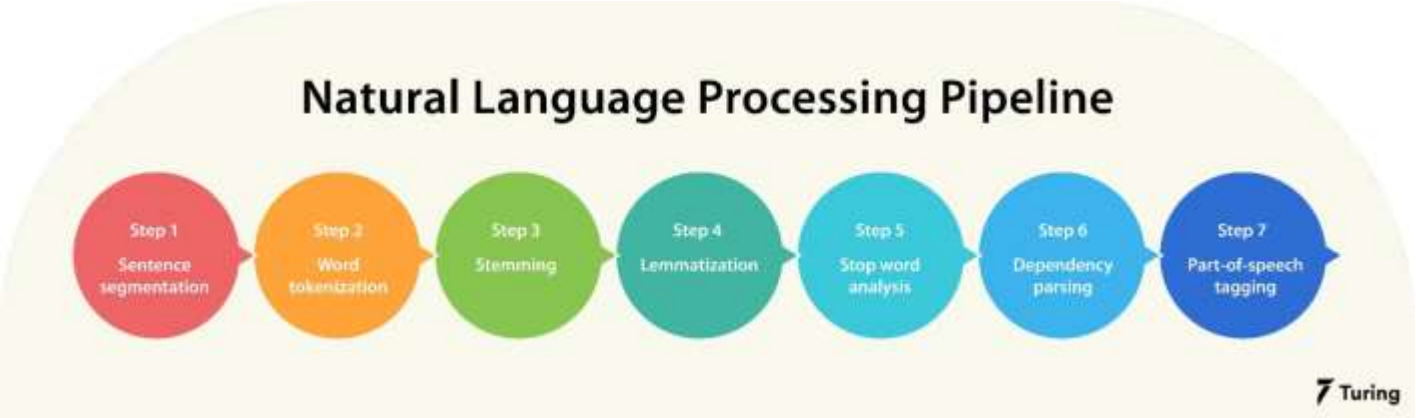
3) **Semantic Analysis**: Semantic analysis is a structure created by the syntactic analyzer which assigns meanings. This component transfers linear sequences of words into structures. It shows how the words are associated with each other. Semantics focuses only on the literal meaning of words, phrases, and sentences. This only draws the dictionary meaning or the real meaning from the given text. The structures assigned by the syntactic analyzer always have assigned meaning The text is checked for meaningfulness. It is done by mapping syntactic structure and objects in the task domain. E.g. "colorless green idea". This would be rejected by the Symantec analysis as colorless here; green doesn't make any sense.

4) **Discourse Integration**: The meaning of any sentence depends upon the meaning of the sentence just before it. Furthermore, it also brings about the meaning of immediately succeeding sentence. For example, "He wanted that", in this sentence the word "that" depends upon the prior discourse context.

5) **Pragmatic Analysis**: Pragmatic analysis concerned with the overall communicative and social content and its effect on interpretation. It means abstracting or deriving the meaningful use of language in situations. In this analysis, what was said is reinterpreted on what it truly meant. It contains deriving those aspects of language which necessitate real world knowledge. E.g., "close the window?" should be interpreted as a request instead of an order.

Sentence Analysis phases

Pipeline of natural language processing in artificial intelligence



The NLP pipeline comprises a set of steps to read and understand human language.

Step 1: Sentence segmentation

Sentence segmentation is the first step in the NLP pipeline. It divides the entire paragraph into different sentences for better understanding. For example, *"London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the southeast of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium."*

Source: Wikipedia

After using sentence segmentation, we get the following result:

1. "London is the capital and most populous city of England and the United Kingdom."
2. "Standing on the River Thames in the southeast of the island of Great Britain, London has been a major settlement for two millennia."
3. "It was founded by the Romans, who named it Londinium."

Step 2: Word tokenization

Word tokenization breaks the sentence into separate words or tokens. This helps understand the context of the text. When tokenizing the sentence "London is the capital and most populous city of England and the United Kingdom", it is broken into separate words, i.e., "London", "is", "the", "capital", "and", "most", "populous", "city", "of", "England", "and", "the", "United", "Kingdom", "."

Step 3: Stemming

Stemming helps in preprocessing text. The model analyzes the parts of speech to figure out what exactly the sentence is talking about.

Stemming normalizes words into their base or root form. In other words, it helps to predict the parts of speech for each token. For example, *intelligently, intelligence, and intelligent*. These words originate from a single root word 'intelligen'. However, in English there's no such word as 'intelligen'.

Image source: Medium.com

The result will be:

Image source: Medium.com

Step 4: Lemmatization

Lemmatization removes inflectional endings and returns the canonical form of a word or lemma. It is similar to stemming except that the lemma is an actual word. For example, 'playing' and 'plays' are forms of the word 'play'. Hence, play is the lemma of these words. Unlike a stem (recall 'intelligen'), 'play' is a proper word.

Step 5: Stop word analysis

The next step is to consider the importance of each and every word in a given sentence. In English, some words appear more frequently than others such as "is", "a", "the", "and". As they appear often, the NLP pipeline flags them as stop words. They are filtered out so as to focus on more important words.

Step 6: Dependency parsing

Next comes dependency parsing which is mainly used to find out how all the words in a sentence are related to each other. To find the dependency, we can build a tree and assign a single word as a parent word. The main verb in the sentence will act as the root node.

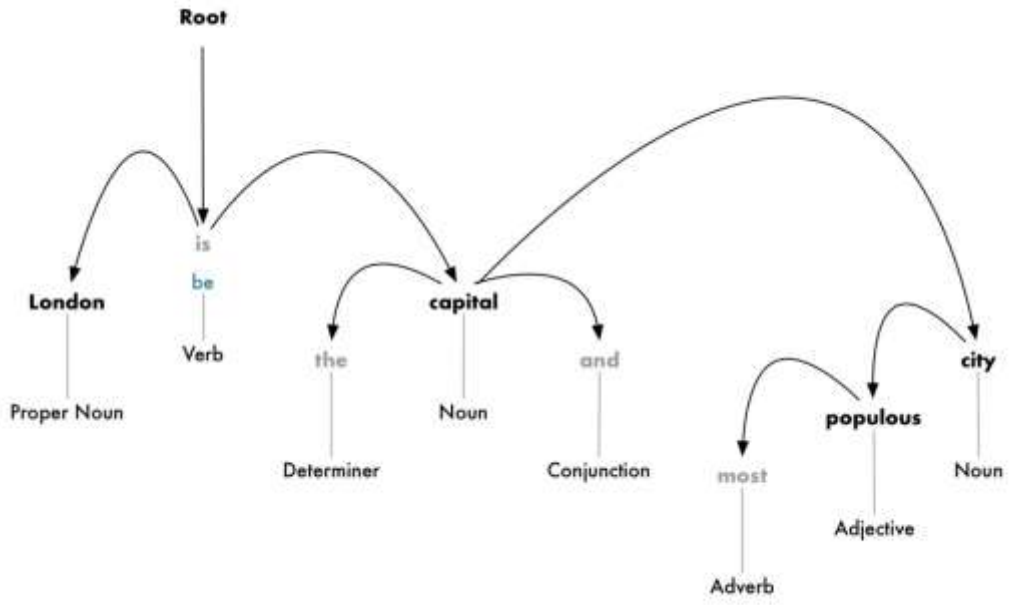


Image source: Medium.com

Step 7: Part-of-speech (POS) tagging

POS tags contain verbs, adverbs, nouns, and adjectives that help indicate the meaning of words in a grammatically correct way in a sentence.

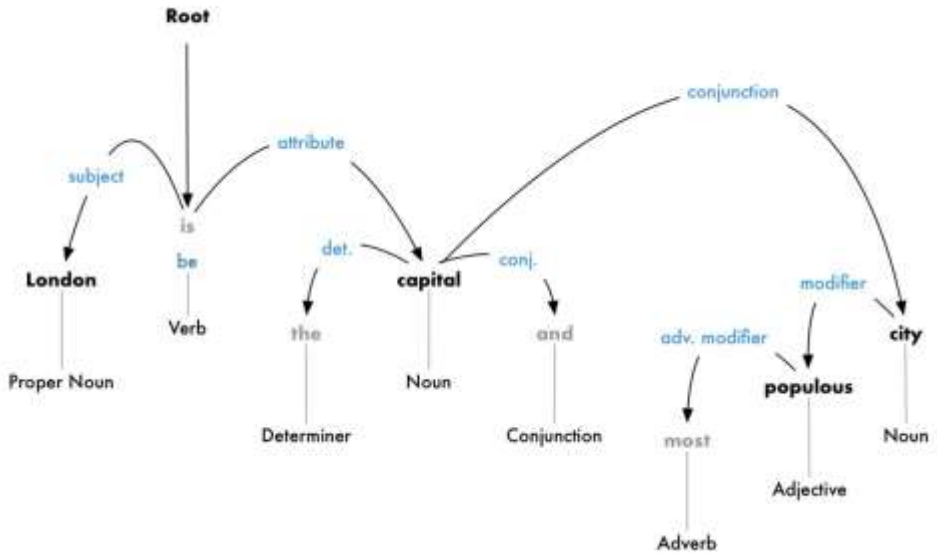


Image source: Medium.com

Natural language processing is used when we want machines to interpret human language. The main goal is to make meaning out of text in order to perform certain tasks automatically such as spell check, translation, for social media monitoring tools, and so on.

Syntactic analysis:

This method examines the structure of a sentence and performs detailed analysis of the sentence and semantics of the statement. In order to perform this, the system is expected to have through knowledge of the grammar of the language. The basic unit of any language is sentence, made up of group of words, having their own meanings and linked together to present an idea or thought. Apart from having meanings, words fall under categories called parts of speech. In English languages, there are eight different parts of speech. They are nouns, pronoun, adjectives, verb, adverb, prepositions, conjunction and interjections.

In English language, a sentence S is made up of a noun phrase (NP) and a verb phrase (VP), i.e.

S=NP+VP

The given noun phrase (NP) normally can have an article or delimiter (D) or an adjective (ADJ) and the noun (N), i.e.

NP=D+ADJ+N

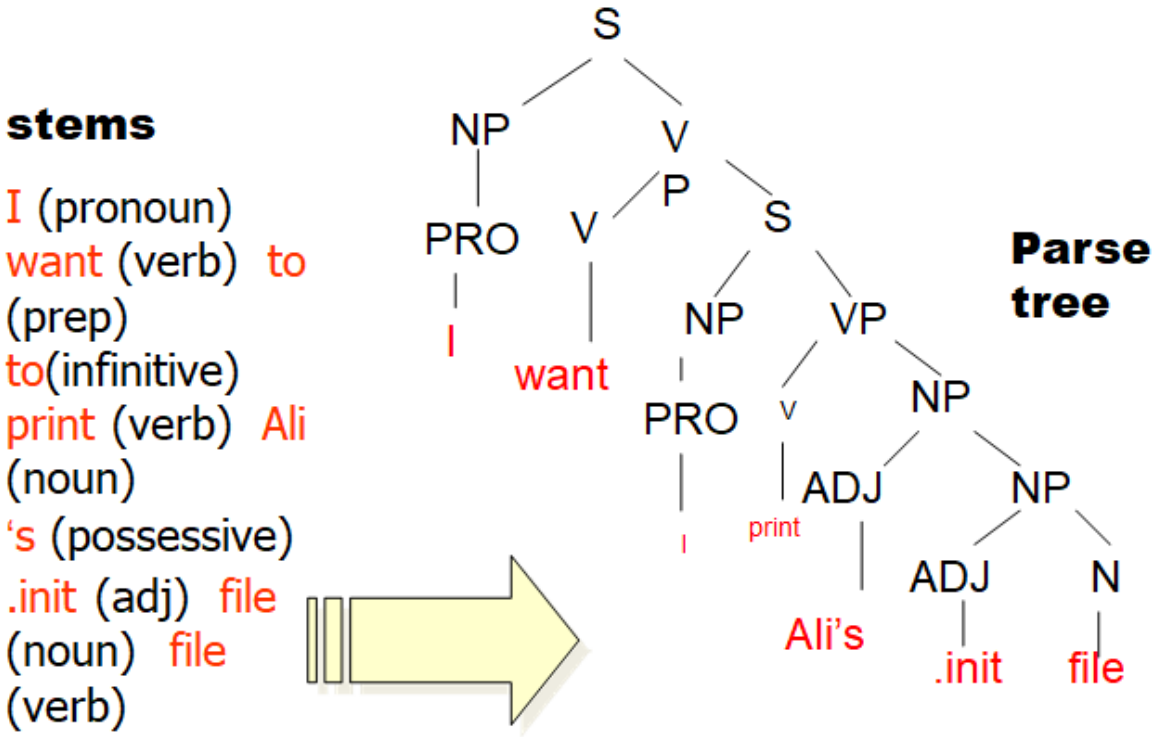
Also a noun phrase may have a prepositional phrase (PP) which has a preposition (P), a delimiter (D) and the noun (N), i.e.

PP=D+P+N

The verb phrase (VP) has a verb (V) and the object of the verb. The object of the verb may be a noun (N) and its determiner, i.e.

VP=V+N+D

These are some of the rules of the English grammar that helps one to construct a small parser for NLP.



Syntactic analysis must exploit the results of morphological analysis to build a structural description of the sentence. The goal of this process, called parsing, is to convert the flat list of words that forms the sentence into a structure that defines the units that are represented by that flat list. The important thing here is that a flat sentence has been converted into a hierarchical structure and that the structures correspond to meaning units 15

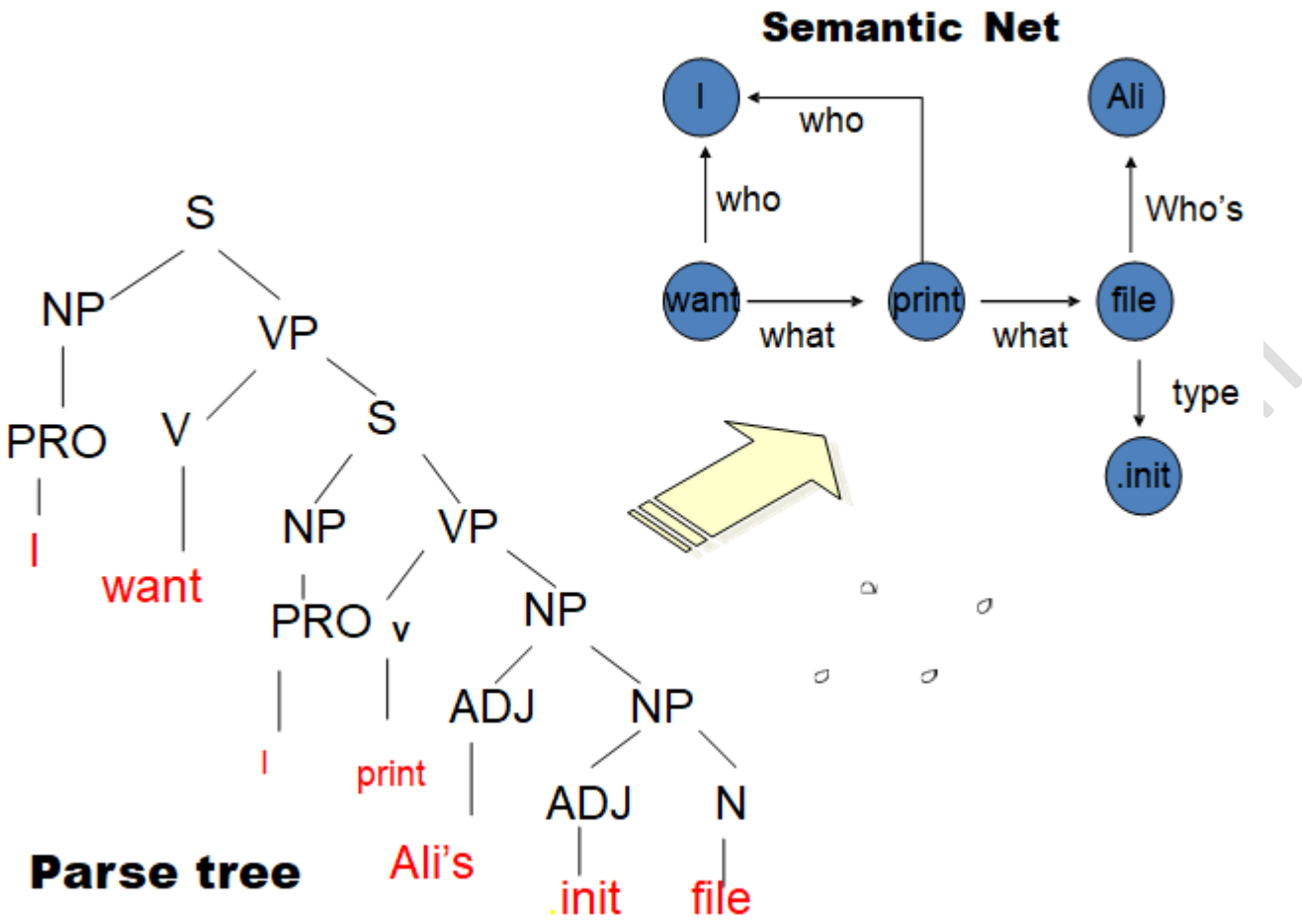
Introduction

when semantic analysis is performed. Reference markers are shown in the parenthesis in the parse tree. Each one corresponds to some entity that has been mentioned in the sentence.

Semantic Analysis:

The structures created by the syntactic analyzer assigned meanings. Also, a mapping made between the syntactic structures and objects in the task domain. Moreover, Structures for which no such mapping possible may reject

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



Important Syntax Analyser Terminology

Important terminologies used in syntax analysis process:

• **Sentence:** A sentence is a group of character over some alphabet.

• **Lexeme:** A lexeme is the lowest level syntactic unit of a language (e.g., total, start).

• **Token:** A token is just a category of lexemes.

• **Keywords and reserved words** – It is an identifier which is used as a fixed part of the syntax of a statement. It is a reserved word which you can't use as a variable name or identifier.

• **Noise words** – Noise words are optional which are inserted in a statement to enhance the readability of the sentence.

• **Comments** – It is a very important part of the documentation. It mostly display by, /* */, or//Blank (spaces)

• **Delimiters** – It is a syntactic element which marks the start or end of some syntactic unit. Like a statement or expression, "begin"…"end", or {}.

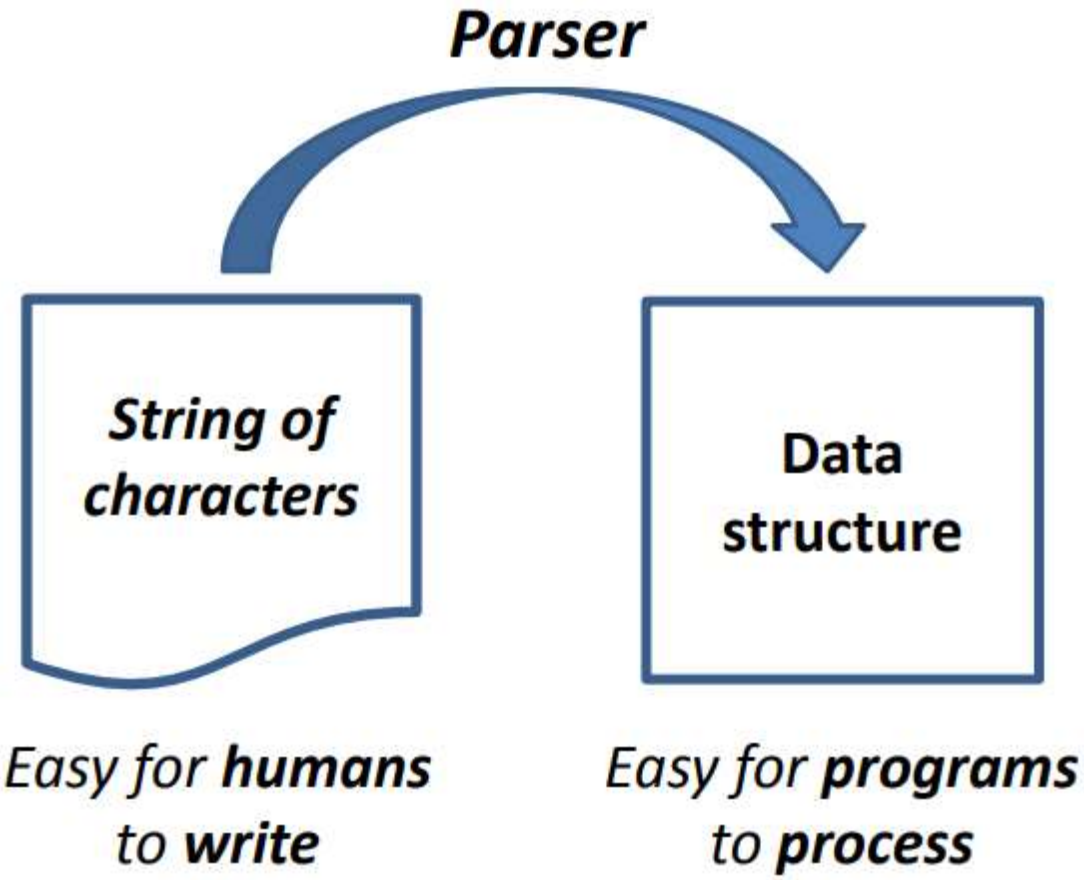
• **Character set** – ASCII, Unicode

• **Identifiers** – It is a restrictions on the length which helps you to reduce the readability of the sentence.

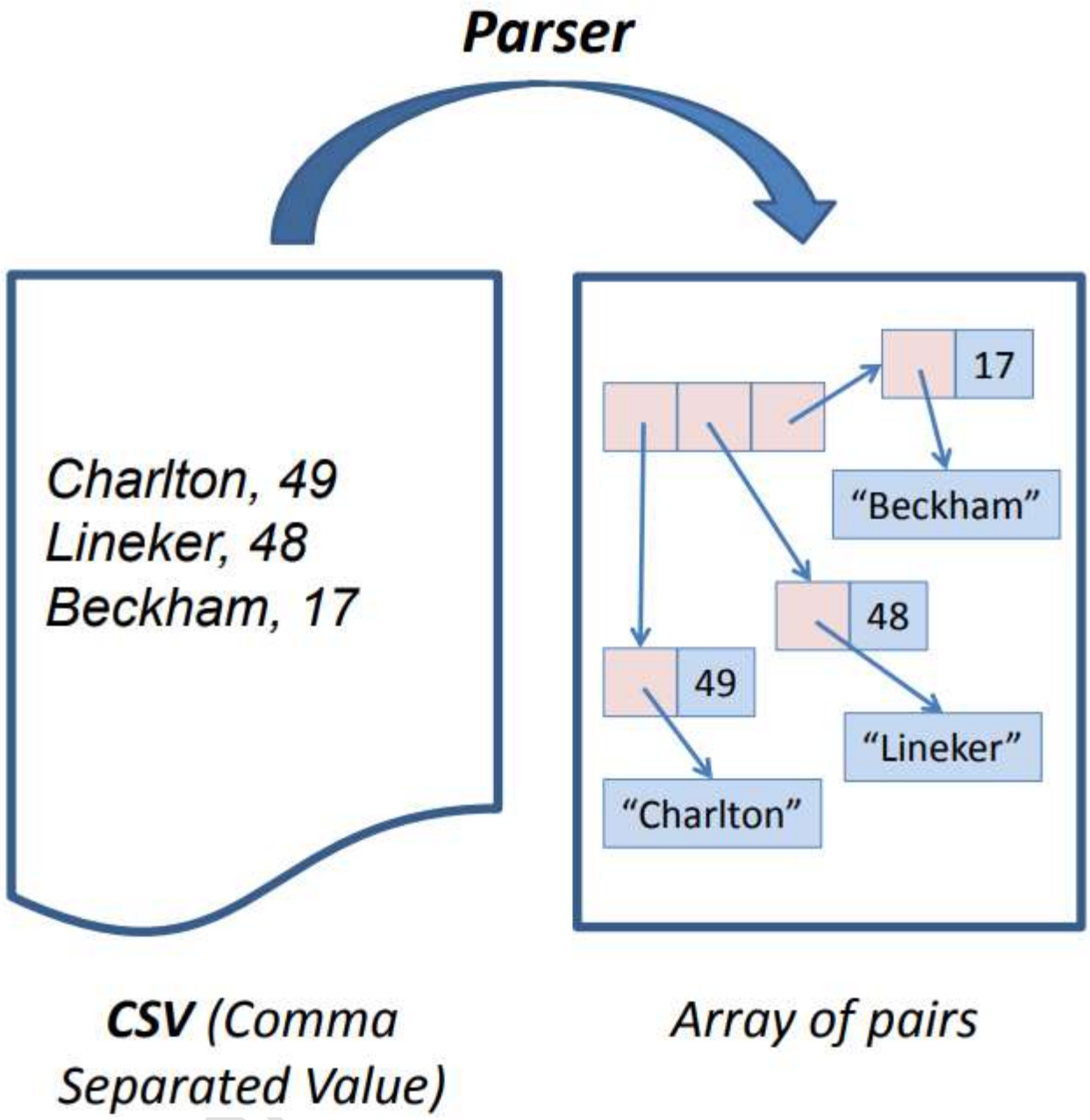
• **Operator symbols** – + and – performs two basic arithmetic operations.

- Syntactic elements of the Language

Why do we need Parsing?



A parse also checks that the input string is well-formed, and if not, reject it.



Following are important tasks perform by the parser in compiler design:

- Helps you to detect all types of Syntax errors
- Find the position at which error has occurred
- Clear & accurate description of the error.
- Recovery from an error to continue and find further errors in the code.
- Should not affect compilation of "correct" programs.
- The parse must reject invalid texts by reporting syntax errors

Parsing Techniques

Parsing techniques are divided into two different groups:

• Top-Down Parsing,

• Bottom-Up Parsing

Top-Down Parsing:

In the top-down parsing construction of the parse tree starts at the root and then proceeds towards the leaves.

Two types of Top-down parsing are:

1. Predictive Parsing:

Predictive parse can predict which production should be used to replace the specific input string. The predictive parser uses look-ahead point, which points towards next input symbols. Backtracking is not an issue with this parsing technique. It is known as LL(1) Parser

2. Recursive Descent Parsing:

This parsing technique recursively parses the input to make a prase tree. It consists of several small functions, one for each nonterminal in the grammar.

Bottom-Up Parsing:

In the bottom up parsing in compiler design, the construction of the parse tree starts with the leave, and then it processes towards its root. It is also called as shift-reduce parsing. This type of parsing in compiler design is created with the help of using some software tools.

Error – Recovery Methods

Common Errors that occur in Parsing in System Software

• **Lexical**: Name of an incorrectly typed identifier

• **Syntactical**: unbalanced parenthesis or a missing semicolon

• **Semantical**: incompatible value assignment

• **Logical**: Infinite loop and not reachable code

A parser should able to detect and report any error found in the program. So, whenever an error occurred the parser. It should be able to handle it and carry on parsing the remaining input. A program can have following types of errors at various compilation process stages. There are five common error-recovery methods which can be implemented in the parser

Statement mode recovery

• In the case when the parser encounters an error, it helps you to take corrective steps. This allows rest of inputs and states to parse ahead.

• For example, adding a missing semicolon is comes in statement mode recover method. However, parse designer need to be careful while making these changes as one wrong correction may lead to an infinite loop.

Panic-Mode recovery

• In the case when the parser encounters an error, this mode ignores the rest of the statement and not process input from erroneous input to delimiter, like a semi-colon. This is a simple error recovery method.

• In this type of recovery method, the parser rejects input symbols one by one until a single designated group of synchronizing tokens is found. The synchronizing tokens generally using delimiters like or.

Phrase-Level Recovery:

• Compiler corrects the program by inserting or deleting tokens. This allows it to proceed to parse from where it was. It performs correction on the remaining input. It can replace a prefix of the remaining input with some string this helps the parser to continue the process.

Error Productions

• Error production recovery expands the grammar for the language which generates the erroneous constructs. The parser then performs error diagnostic about that construct.

Global Correction:

• The compiler should make less number of changes as possible while processing an incorrect input string. Given incorrect input string a and grammar c, algorithms will search for a parse tree for a related string b. Like some insertions, deletions, and modification made of tokens needed to transform an into b is as little as possible.

Grammar:

A grammar is a set of structural rules which describe a language. Grammars assign structure to any sentence. This term also refers to the study of these rules, and this file includes morphology, phonology, and syntax. It is capable of describing many, of the syntax of programming languages. 61

Syntax Analysis - I

Rules of Form Grammar

• The non-terminal symbol should appear to the left of the at least one production

• The goal symbol should never be displayed to the right of the::= of any production

• A rule is recursive if LHS appears in its RHS

Notational Conventions

Notational conventions symbol may be indicated by enclosing the element in square brackets. It is an arbitrary sequence of instances of the element which can be indicated by enclosing the element in braces followed by an asterisk symbol, { … }*.

It is a choice of the alternative which may use the symbol within the single rule. It may be enclosed by parenthesis ([,]) when needed.

Two types of Notational conventions area Terminal and Non-terminals

1. Terminals:

- Lower-case letters in the alphabet such as a, b, c,
- Operator symbols such as +,-, *, etc.
- Punctuation symbols such as parentheses, hash, comma
- 0, 1, …, 9 digits
- Boldface strings like id or if, anything which represents a single terminal symbol

2.Nonterminals:

- Upper-case letters such as A, B, C
- Lower-case italic names: the expression or some

Context Free Grammar

A CFG is a left-recursive grammar that has at least one production of the type. The rules in a context-free grammar are mainly recursive. A syntax analyser checks that specific program satisfies all the rules of Context-free grammar or not. If it does meet, these rules syntax analysers may create a parse tree for that programme.

expression -> expression -+ term

expression -> expression – term

expression-> term

term -> term * factor

term -> expression/ factor

term -> factor factor

factor -> (expression)

factor -> id

Grammar Derivation

Grammar derivation is a sequence of grammar rule which transforms the start symbol into the string. A derivation proves that the string belongs to the grammar's language.

Left-most Derivation

When the sentential form of input is scanned and replaced in left to right sequence, it is known as left-most derivation. The sentential form which is derived by the left-most derivation is called the left-sentential form.

Right-most Derivation

Rightmost derivation scan and replace the input with production rules, from right to left, sequence. It's known as right-most derivation. The sentential form which is derived from the rightmost derivation is known as right-sentential form.

Syntax vs. Lexical Analyser

Syntax Analyser

The syntax analyser mainly deals with recursive constructs of the language.

The syntax analyser works on tokens in a source program to recognize meaningful structures in the programming language.

It receives inputs, in the form of tokens, from lexical analysers.

Lexical Analyser

The lexical analyser eases the task of the syntax analyser.

The lexical analyser recognizes the token in a source program.

It is responsible for the validity of a token supplied by the syntax analyser

Disadvantages of using Syntax Analysers

- It will never determine if a token is valid or not
- Not helps you to determine if an operation performed on a token type is valid or not
- You can't decide that token is declared & initialized before it is being used

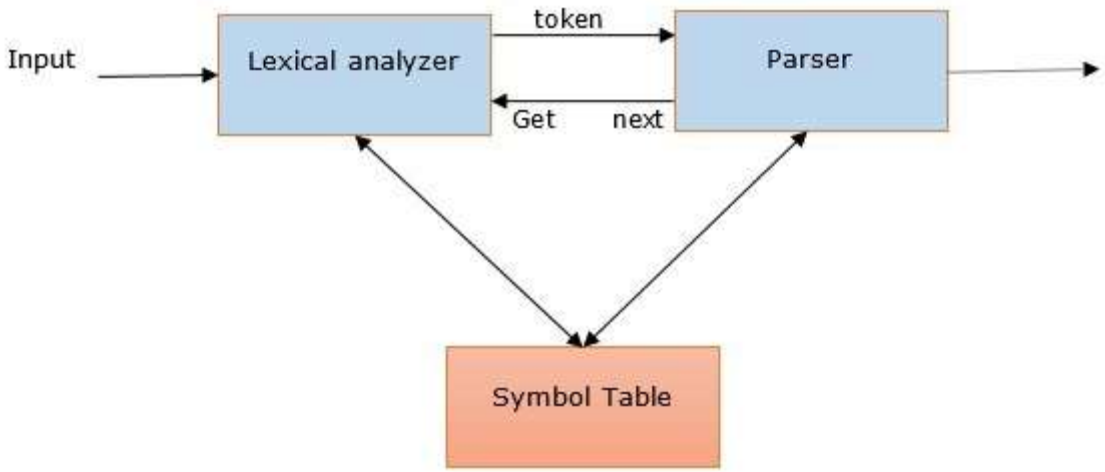
---

Parsing and its relevance in NLP

The word 'Parsing' whose origin is from Latin word **'pars'** (which means **'part'**), is used to draw exact meaning or dictionary meaning from the text. It is also called Syntactic analysis or syntax analysis. Comparing the rules of formal grammar, syntax analysis checks the text for meaningfulness. The sentence like "Give me hot ice-cream", for example, would be rejected by parser or syntactic analyzer.

In this sense, we can define parsing or syntactic analysis or syntax analysis as follows –

It may be defined as the process of analyzing the strings of symbols in natural language conforming to the rules of formal grammar.



We can understand the relevance of parsing in NLP with the help of following points –

- Parser is used to report any syntax error.
- It helps to recover from commonly occurring error so that the processing of the remainder of program can be continued.
- Parse tree is created with the help of a parser.
- Parser is used to create symbol table, which plays an important role in NLP.
- Parser is also used to produce intermediate representations (IR).

Deep Vs Shallow Parsing

Deep Parsing	**Shallow Parsing**
In deep parsing, the search strategy will give a complete syntactic structure to a sentence.	It is the task of parsing a limited part of the syntactic information from the given task.
It is suitable for complex NLP applications.	It can be used for less complex NLP applications.
Dialogue systems and summarization are the examples of NLP applications where deep parsing is used.	Information extraction and text mining are the examples of NLP applications where deep parsing is used.
It is also called full parsing.	It is also called chunking.

Various types of parsers

As discussed, a parser is basically a procedural interpretation of grammar. It finds an optimal tree for the given sentence after searching through the space of a variety of trees. Let us see some of the available parsers below –

Recursive descent parser

Recursive descent parsing is one of the most straightforward forms of parsing. Following are some important points about recursive descent parser –

- It follows a top down process.
- It attempts to verify that the syntax of the input stream is correct or not.
- It reads the input sentence from left to right.
- One necessary operation for recursive descent parser is to read characters from the input stream and matching them with the terminals from the grammar.

Shift-reduce parser

Following are some important points about shift-reduce parser –

- It follows a simple bottom-up process.
- It tries to find a sequence of words and phrases that correspond to the right-hand side of a grammar production and replaces them with the left-hand side of the production.
- The above attempt to find a sequence of word continues until the whole sentence is reduced.
- In other simple words, shift-reduce parser starts with the input symbol and tries to construct the parser tree up to the start symbol.

Chart parser

Following are some important points about chart parser –

- It is mainly useful or suitable for ambiguous grammars, including grammars of natural languages.
- It applies dynamic programing to the parsing problems.
- Because of dynamic programing, partial hypothesized results are stored in a structure called a 'chart'.
- The 'chart' can also be re-used.

Regexp parser

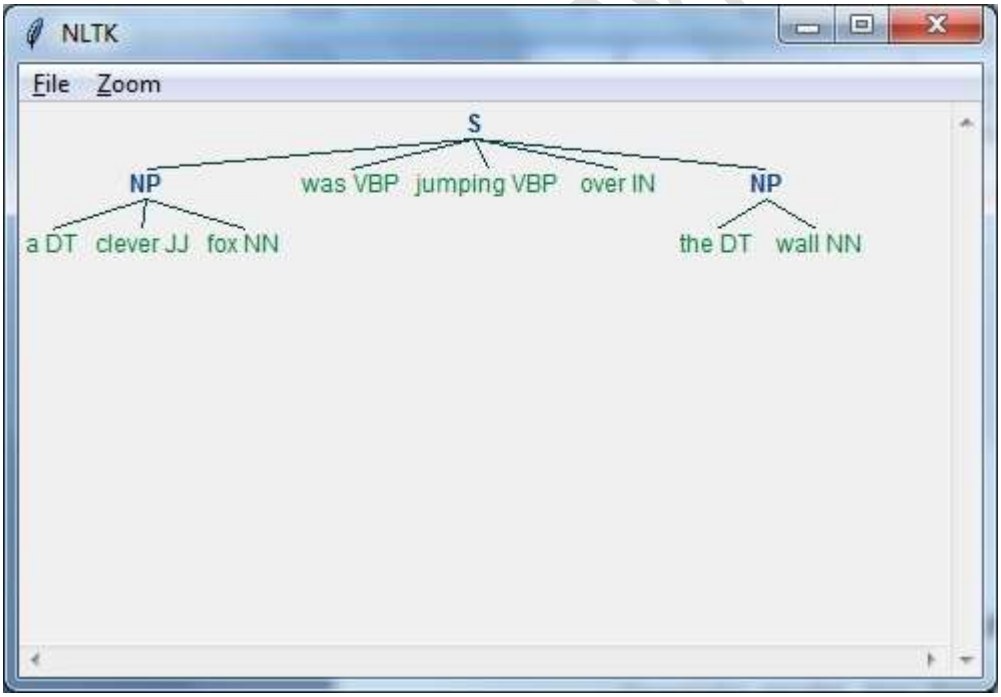
Regexp parsing is one of the mostly used parsing technique. Following are some important points about Regexp parser –

- As the name implies, it uses a regular expression defined in the form of grammar on top of a POS-tagged string.
- It basically uses these regular expressions to parse the input sentences and generate a parse tree out of this.

Example

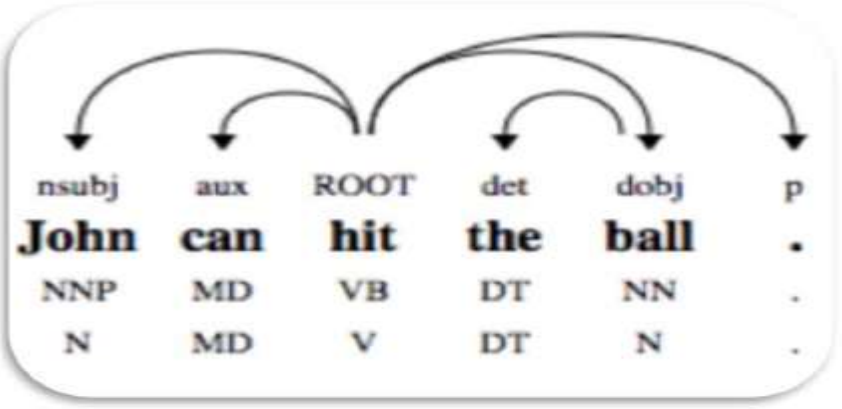
Following is a working example of Regexp Parser –

```
import nltk
sentence = [
   ("a", "DT"),
   ("clever", "JJ"),
   ("fox","NN"),
   ("was","VBP"),
   ("jumping","VBP"),
   ("over","IN"),
   ("the","DT"),
   ("wall","NN")
]
grammar = "NP:{<DT>?<JJ>*<NN>}"
Reg_parser = nltk.RegexpParser(grammar)
Reg_parser.parse(sentence)
Output = Reg_parser.parse(sentence)
Output.draw()
Output
```



Dependency Parsing

Dependency Parsing (DP), a modern parsing mechanism, whose main concept is that each linguistic unit i.e. words relates to each other by a direct link. These direct links are actually ***'dependencies'*** in linguistic. For example, the following diagram shows dependency grammar for the sentence ***"John can hit the ball"***.



NLTK Package

We have following the two ways to do dependency parsing with NLTK –

Probabilistic, projective dependency parser

This is the first way we can do dependency parsing with NLTK. But this parser has the restriction of training with a limited set of training data.

Stanford parser

This is another way we can do dependency parsing with NLTK. Stanford parser is a state-of-the-art dependency parser. NLTK has a wrapper around it. To use it we need to download following two things

The Stanford CoreNLP parser.

Language model for desired language. For example, English language model.

Example

Once you downloaded the model, we can use it through NLTK as follows –

```
from nltk.parse.stanford import StanfordDependencyParser

path_jar = 'path_to/stanford-parser-full-2014-08-27/stanford-parser.jar'

path_models_jar = 'path_to/stanford-parser-full-2014-08-27/stanford-parser-3.4.1-models.jar'

dep_parser = StanfordDependencyParser(

   path_to_jar = path_jar, path_to_models_jar = path_models_jar

)

result = dep_parser.raw_parse('I shot an elephant in my sleep')

depndency = result.next()

list(dependency.triples())
```

Output

```
[

   ((u'shot', u'VBD'), u'nsubj', (u'I', u'PRP')),

   ((u'shot', u'VBD'), u'dobj', (u'elephant', u'NN')),
```

```
    ((u'elephant', u'NN'), u'det', (u'an', u'DT')),
    ((u'shot', u'VBD'), u'prep', (u'in', u'IN')),
    ((u'in', u'IN'), u'pobj', (u'sleep', u'NN')),
    ((u'sleep', u'NN'), u'poss', (u'my', u'PRP$'))
]
```

Introduction to Semantic Analysis

Semantic Analysis is a subfield of Natural Language Processing (NLP) that attempts to understand the meaning of Natural Language. Understanding Natural Language might seem a straightforward process to us as humans. However, due to the vast complexity and subjectivity involved in human language, interpreting it is quite a complicated task for machines. Semantic Analysis of Natural Language captures the meaning of the given text while taking into account context, logical structuring of sentences and grammar roles.

Parts of Semantic Analysis

Semantic Analysis of Natural Language can be classified into two broad parts:

1. Lexical Semantic Analysis: Lexical Semantic Analysis involves understanding the meaning of each word of the text individually. It basically refers to fetching the dictionary meaning that a word in the text is deputed to carry.

2. Compositional Semantics Analysis: Although knowing the meaning of each word of the text is essential, it is not sufficient to completely understand the meaning of the text.

For example, consider the following two sentences:

- **Sentence 1:** Students love GeeksforGeeks.
- **Sentence 2:** GeeksforGeeks loves Students.

Although both these sentences 1 and 2 use the same set of root words {student, love, geeksforgeeks}, they convey entirely different meanings.

Hence, under Compositional Semantics Analysis, we try to understand how combinations of individual words form the meaning of the text.

Tasks involved in Semantic Analysis

In order to understand the meaning of a sentence, the following are the major processes involved in Semantic Analysis:

1. Word Sense Disambiguation
2. Relationship Extraction

Word Sense Disambiguation:

In Natural Language, the meaning of a word may vary as per its usage in sentences and the context of the text. Word Sense Disambiguation involves interpreting the meaning of a word based upon the context of its occurrence in a text.

For example, the word 'Bark' may mean 'the sound made by a dog' or 'the outermost layer of a tree.'

Likewise, the word 'rock' may mean '*a stone*' or '*a genre of music*' – hence, the accurate meaning of the word is highly dependent upon its context and usage in the text.

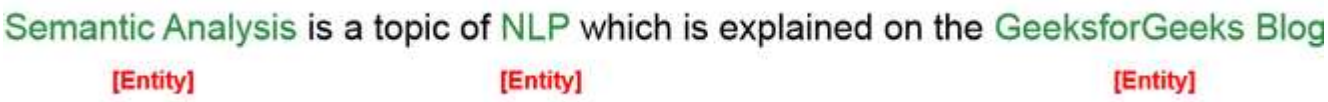
Thus, the ability of a machine to overcome the ambiguity involved in identifying the meaning of a word based on its usage and context is called Word Sense Disambiguation.

Relationship Extraction:

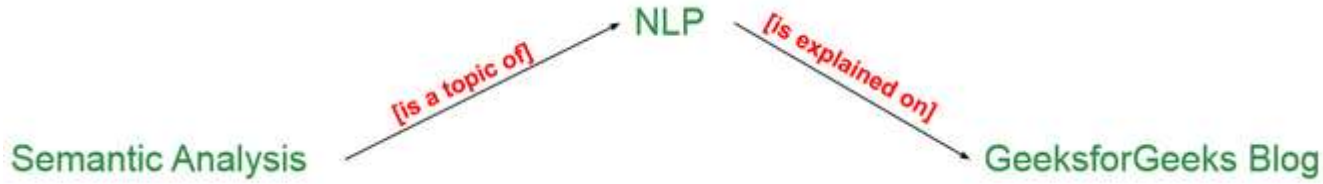
Another important task involved in Semantic Analysis is Relationship Extracting. It involves firstly identifying various entities present in the sentence and then extracting the relationships between those entities.

For example, consider the following sentence:

Semantic Analysis is a topic of NLP which is explained on the GeeksforGeeks blog. The entities involved in this text, along with their relationships, are shown below.



Entities



Relationships

Elements of Semantic Analysis

Some of the critical elements of Semantic Analysis that must be scrutinized and taken into account while processing Natural Language are:

- **Hyponymy:** Hyponymys refers to a term that is an instance of a generic term. They can be understood by taking class-object as an analogy. For example: '*Color*' is a hypernymy while '*grey*', '*blue*', '*red*', etc, are its hyponyms.
- **Homonymy:** Homonymy refers to two or more lexical terms with the same spellings but completely distinct in meaning. For example: '*Rose*' might mean '*the past form of rise*' or '*a flower*', – same spelling but different meanings; hence, '*rose*' is a homonymy.
- **Synonymy:** When two or more lexical terms that might be spelt distinctly have the same or similar meaning, they are called Synonymy. For example: *(Job, Occupation), (Large, Big), (Stop, Halt).*
- **Antonymy:** Antonymy refers to a pair of lexical terms that have contrasting meanings – they are symmetric to a semantic axis. For example: *(Day, Night), (Hot, Cold), (Large, Small).*
- **Polysemy:** Polysemy refers to lexical terms that have the same spelling but multiple closely related meanings. It differs from homonymy because the meanings of the terms need not be closely related in the case of homonymy. For example: '*man*' may mean '*the human species*' or '*a male human*' or '*an adult male human*' – since all these different meanings bear a close association, the lexical term '*man*' is a polysemy.
- **Meronomy:** Meronomy refers to a relationship wherein one lexical term is a constituent of some larger entity. For example: '*Wheel*' is a meronym of '*Automobile*'

Meaning Representation

While, as humans, it is pretty simple for us to understand the meaning of textual information, it is not so in the case of machines. Thus, machines tend to represent the text in specific formats in order to interpret its meaning. This formal structure that is used to understand the meaning of a text is called meaning representation.

Basic Units of Semantic System:

In order to accomplish Meaning Representation in Semantic Analysis, it is vital to understand the building units of such representations. The basic units of semantic systems are explained below:

1. **Entity:** An entity refers to a particular unit or individual in specific such as a person or a location. For example GeeksforGeeks, Delhi, etc.
2. **Concept:** A Concept may be understood as a generalization of entities. It refers to a broad class of individual units. For example Learning Portals, City, Students.
3. **Relations:** Relations help establish relationships between various entities and concepts. For example: 'GeeksforGeeks is a Learning Portal', 'Delhi is a City.', etc.
4. **Predicate:** Predicates represent the verb structures of the sentences.

In Meaning Representation, we employ these basic units to represent textual information.

Approaches to Meaning Representations:

Now that we are familiar with the basic understanding of Meaning Representations, here are some of the most popular approaches to meaning representation:

1. First-order predicate logic (FOPL)
2. Semantic Nets
3. Frames
4. Conceptual dependency (CD)
5. Rule-based architecture
6. Case Grammar
7. Conceptual Graphs

Semantic Analysis Techniques

Based upon the end goal one is trying to accomplish, Semantic Analysis can be used in various ways. Two of the most common Semantic Analysis techniques are:

Text Classification

In-Text Classification, our aim is to label the text according to the insights we intend to gain from the textual data.

For example:

- In **Sentiment Analysis,** we try to label the text with the prominent emotion they convey. It is highly beneficial when analyzing customer reviews for improvement.
- In **Topic Classification**, we try to categories our text into some predefined categories. For example: Identifying whether a research paper is of Physics, Chemistry or Maths

- In **Intent Classification**, we try to determine the intent behind a text message. For example: Identifying whether an e-mail received at customer care service is a query, complaint or request.

Text Extraction

In-Text Extraction, we aim at obtaining specific information from our text.

For Example,

- In **Keyword Extraction**, we try to obtain the essential words that define the entire document.
- In **Entity Extraction**, we try to obtain all the entities involved in a document.

Significance of Semantics Analysis

Semantics Analysis is a crucial part of Natural Language Processing (NLP). In the ever-expanding era of textual information, it is important for organizations to draw insights from such data to fuel businesses. Semantic Analysis helps machines interpret the meaning of texts and extract useful information, thus providing invaluable data while reducing manual efforts.

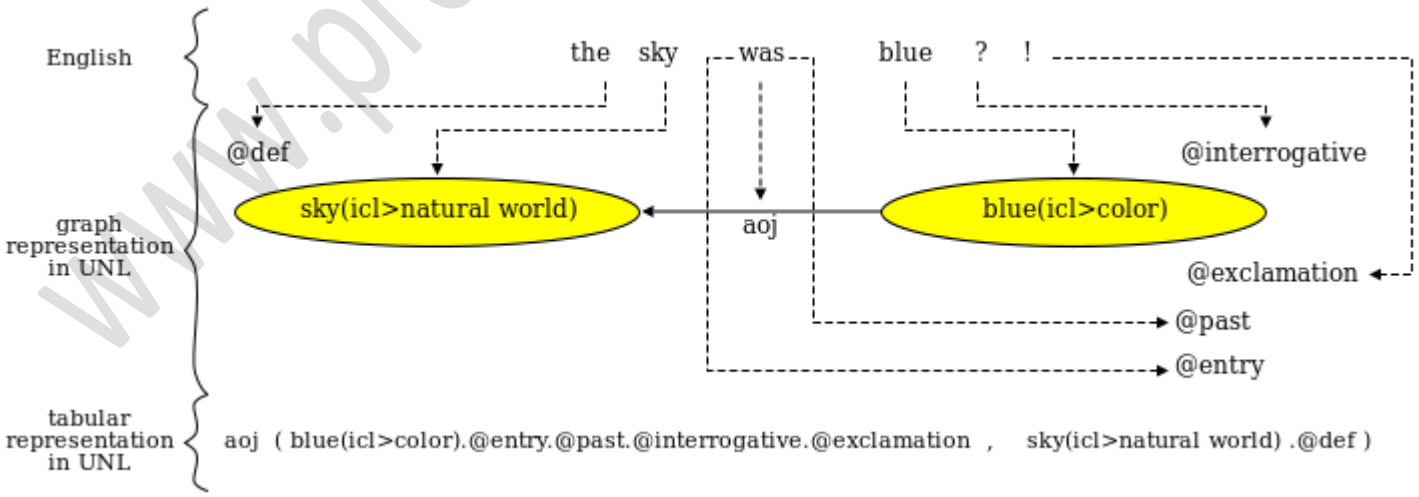
Besides, Semantics Analysis is also widely employed to facilitate the processes of automated answering systems such as chatbots – that answer user queries without any human interventions.

Universal Networking Language (UNL)

Universal Networking Language (UNL) is a declarative formal language specifically designed to represent semantic data extracted from natural language texts. It can be used as a pivot language in interlingual machine translation systems or as a knowledge representation language in information retrieval applications.

In the UNL approach, information conveyed by natural language is represented sentence by sentence as a hypergraph composed of a set of directed binary labeled links (referred to as **relations**) between nodes or hypernodes (the **Universal Words**, or simply **UWs**), which stand for concepts. UWs can also be annotated with **attributes** representing context information.

As an example, the English sentence 'The sky was blue?!' can be represented in UNL as follows:



In the example above, sky(icl>natural world) and blue(icl>color), which represent individual concepts, are UWs; "aoj" (= attribute of an object) is a directed binary semantic relation linking the two UWs; and "@def", "@interrogative", "@past", "@exclamation" and "@entry" are attributes modifying UWs.

UWs are intended to represent universal concepts, but are expressed in English words or in any other natural language in order to be humanly readable. They consist of a "headword" (the UW root) and a "constraint list" (the UW suffix between parentheses), where the constraints are used to disambiguate the general concept conveyed by the headword. The set of UWs is organized in the UNL Ontology, in which high-level concepts are related to lower-level ones through the relations "icl" (= is a kind of), "iof" (= is an instance of) and "equ" (= is equal to).

Relations are intended to represent semantic links between words in every existing language. They can be ontological (such as "icl" and "iof," referred to above), logical (such as "and" and "or"), and thematic (such as "agt" = agent, "ins" = instrument, "tim" = time, "plc" = place, etc.). There are currently 46 relations in the UNL Specs. They jointly define the UNL syntax.

Attributes represent information that cannot be conveyed by UWs and relations. Normally, they represent information concerning time ("@past", "@future", etc.), reference ("@def", "@indef", etc.), modality ("@can", "@must", etc.), focus ("@topic", "@focus", etc.), and so on.

Within the UNL Program, the process of representing natural language sentences in UNL graphs is called **UNLization**, and the process of generating natural language sentences out of UNL graphs is called **NLization**. UNLization, which involves natural language analysis and understanding, is intended to be carried out semi-automatically (i.e., by humans with computer aids); and NLization is intended to be carried out fully automatically.