

D.N.R.COLLEGE(AUTONOMOUS):BHIMAVARAM
M.C.A DEPARTMENT



ARTIFICIAL INTELLIGENCE

Presented by
L.SOWJANYA

UNIT-1

What is AI, The Foundations of AI, The History of AI, Agents and Environments, The Concept of Rationality, The Nature of Environments, The Structure of Agents, Problem Solving Agents, Example Problems, Searching for Solutions,

Uninformed Search Strategies: Breadth First, Depth First, Depth Limited;

Informed Search Strategies: Greedy Best First, A* Algorithms

AI DEFINITION:

According to Elaine Rich,

“Artificial intelligence is the study of how to make computers do things which at the moment people are better.”

According to Bruce Buchanan and Edward Shortliffe,

“ Artificial intelligence is the branch of computer science that deals with symbols and uses non algorithmic methods to solve problems.”

According to Avon Barr and Edward A.Feigenbaum,

“Artificial intelligence is the branch of computer science that deals with design of intelligent systems.”

Approaches of AI:

Historically, four approaches to AI have been followed, each by different people with different methods.

1.Thinking Humanly: Systems that think like humans, The cognitive modelling approach

This approach consists of three ways:

Through introspection--trying to catch our own thoughts as they go by.

Through psychological experiments – observing a person in action and

Through brain imaging – observing the brain in action.

2. Acting Humanly: Systems that act like humans, The Turing Test approach.

In 1950, Alan Turing introduced a test to check whether a machine can think like a human or not, this test is known as the Turing Test. In this test, Turing proposed that the computer can be said to be an intelligent if it can mimic human response under specific conditions.

Turing Test was introduced by Turing in his 1950 paper, "Computing Machinery and Intelligence," which considered the question, "Can Machine think?"

The Turing test is based on a party game "Imitation game," with some modifications. This game involves three players in which one player is Computer, another player is human responder, and the third player is a human Interrogator, who is isolated from other two players and his job is to find that which player is machine among two of them. Consider, Player A is a computer, Player B is human, and Player C is an interrogator. Interrogator is aware that one of them is machine, but he needs to identify this on the basis of questions and their responses.

The conversation between all players is via keyboard and screen so the result would not depend on the machine's ability to convert words as speech.

The test result does not depend on each correct answer, but only how closely its responses like a human answer. The computer is permitted to do everything possible to force a wrong identification by the interrogator.

The questions and answers can be like:

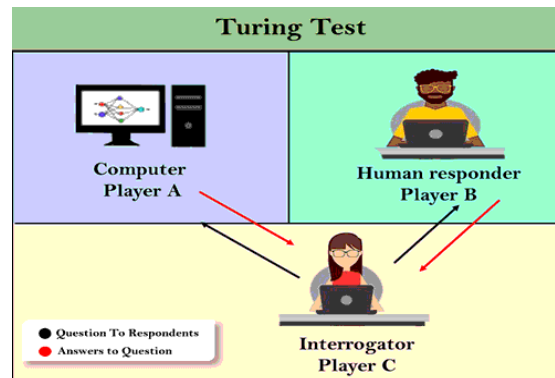
Interrogator: Are you a computer?

Player A (Computer): No

Interrogator: Multiply two large numbers such as (256896489*456725896)

Player A: Long pause and give the wrong answer.

In this game, if an interrogator would not be able to identify which is a machine and which is human, then the computer passes the test successfully, and the machine is said to be intelligent and can think like a human.



The computer would need to possess the following capabilities:

- **Natural language processing** to enable it to communicate successfully in English (or some other human language);
- **Knowledge representation** to store information provided before or during the interrogation;
- **Automated reasoning** to use the stored information to answer questions and to draw new conclusions;
- **Machine learning** to adapt to new circumstances and to detect and extrapolate patterns.

3. Thinking Rationally: Systems that think rationally, The laws of thought approach

The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking." His famous syllogisms provided patterns for argument structures that always gave correct conclusions given correct premises. For example, "Socrates is a man; all men are mortal; therefore Socrates is mortal." These laws of thought were supposed to govern the operation of the mind, and initiated the field of logic.

There are two main obstacles to this approach. First, it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain. Second, there is a big difference between being able to solve a problem "in principle" and doing so in practice.

4. Acting rationally: Systems that act rationally, The rational agent approach

Acting rationally means acting so as to achieve one's goals, given one's beliefs. An **agent** is just something that perceives and acts. In this approach, AI is viewed as the study and construction of rational agents. A Rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

Why Artificial Intelligence?

Before Learning about Artificial Intelligence, we should know that what is the importance of AI and why should we learn it. Following are some main reasons to learn about AI:

- With the help of AI, you can create such software or devices which can solve real-world problems very easily and with accuracy such as health issues, marketing, traffic issues, etc.

- you can create your personal virtual Assistant, such as Cortana, Google Assistant, Siri, etc.
- you can build such Robots which can work in an environment where survival of humans can be at risk.
- AI opens a path for other new technologies, new devices, and new Opportunities.

Goals of Artificial Intelligence

Following are the main goals of Artificial Intelligence:

- Replicate human intelligence
- Solve Knowledge-intensive tasks
- An intelligent connection of perception and action
- Building a machine which can perform tasks that requires human intelligence such as:
 - Proving a theorem
 - Playing chess
 - Plan some surgical operation
 - Driving a car in traffic
- Creating some system which can exhibit intelligent behavior, learn new things by itself, demonstrate, explain, and can advise to its user.

Advantages of Artificial Intelligence

Following are some main advantages of Artificial Intelligence:

- **High Accuracy with fewer errors:** AI machines or systems are prone to less errors and high accuracy as it takes decisions as per pre-experience or information.
- **High-Speed:** AI systems can be of very high-speed and fast-decision making, because of that AI systems can beat a chess champion in the Chess game.
- **High reliability:** AI machines are highly reliable and can perform the same action multiple times with high accuracy.
- **Useful for risky areas:** AI machines can be helpful in situations such as defusing a bomb, exploring the ocean floor, where to employ a human can be risky.
- **Digital Assistant:** AI can be very useful to provide digital assistant to the users such as AI technology is currently used by various E-commerce websites to show the products as per customer requirement.
- **Useful as a public utility:** AI can be very useful for public utilities such as a self-driving car which can make our journey safer and hassle-free, facial recognition for security purpose, Natural language processing to communicate with the human in human-language, etc.

Disadvantages of Artificial Intelligence

Every technology has some disadvantages, and the same goes for Artificial intelligence. Being so advantageous technology still, it has some disadvantages which we need to keep in our mind while creating an AI system. Following are the disadvantages of AI:

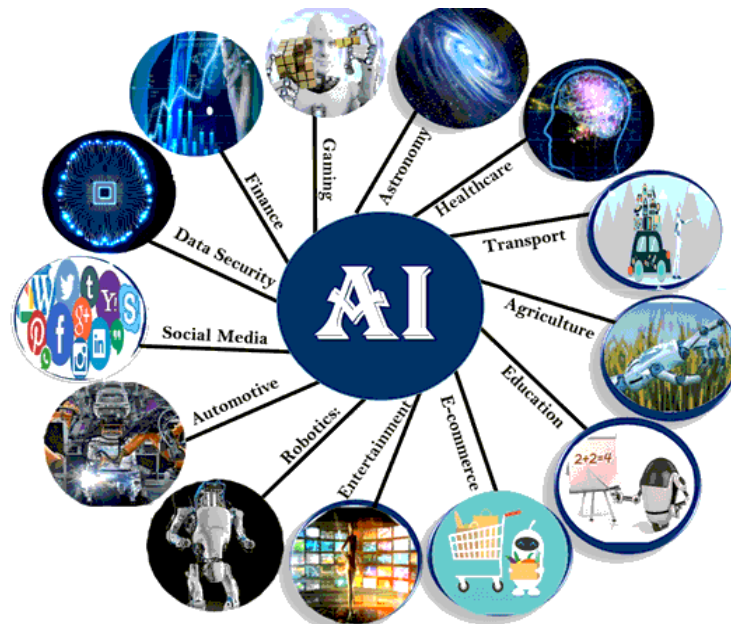
- **High Cost:** The hardware and software requirement of AI is very costly as it requires lots of maintenance to meet current world requirements.

- **Can't think out of the box:** Even we are making smarter machines with AI, but still they cannot work out of the box, as the robot will only do that work for which they are trained, or programmed.
- **No feelings and emotions:** AI machines can be an outstanding performer, but still it does not have the feeling so it cannot make any kind of emotional attachment with human, and may sometime be harmful for users if the proper care is not taken.
- **Increase dependency on machines:** With the increment of technology, people are getting more dependent on devices and hence they are losing their mental capabilities.
- **No Original Creativity:** As humans are so creative and can imagine some new ideas but still AI machines cannot beat this power of human intelligence and cannot be creative and imaginative.

Applications of AI:

Artificial Intelligence has various applications in today's society. It is becoming essential for today's time because it can solve complex problems with an efficient way in multiple industries, such as Healthcare, entertainment, finance, education, etc. AI is making our daily life more comfortable and fast.

Following are some sectors which have the application of Artificial Intelligence:



1. AI in Astronomy

- Artificial Intelligence can be very useful to solve complex universe problems. AI technology can be helpful for understanding the universe such as how it works, origin, etc.

2. AI in Healthcare

- In the last, five to ten years, AI becoming more advantageous for the healthcare industry and going to have a significant impact on this industry.
- Healthcare Industries are applying AI to make a better and faster diagnosis than humans. AI can help doctors with diagnoses and can inform when patients are worsening so that medical help can reach to the patient before hospitalization.

3. AI in Gaming

- AI can be used for gaming purpose. The AI machines can play strategic games like chess, where the machine needs to think of a large number of possible places.

4. AI in Finance

- AI and finance industries are the best matches for each other. The finance industry is implementing automation, chatbot, adaptive intelligence, algorithm trading, and machine learning into financial processes.

5. AI in Data Security

- The security of data is crucial for every company and cyber-attacks are growing very rapidly in the digital world. AI can be used to make your data more safe and secure. Some examples such as AEG bot, AI2 Platform, are used to determine software bug and cyber-attacks in a better way.

6. AI in Social Media

- Social Media sites such as Facebook, Twitter, and Snapchat contain billions of user profiles, which need to be stored and managed in a very efficient way. AI can organize and manage massive amounts of data. AI can analyze lots of data to identify the latest trends, hashtag, and requirement of different users.

7. AI in Travel & Transport

- AI is becoming highly demanding for travel industries. AI is capable of doing various travel related works such as from making travel arrangement to suggesting the hotels, flights, and best routes to the customers. Travel industries are using AI-powered chatbots which can make human-like interaction with customers for better and fast response.

8. AI in Automotive Industry

- Some Automotive industries are using AI to provide virtual assistant to their user for better performance. Such as Tesla has introduced TeslaBot, an intelligent virtual assistant.
- Various Industries are currently working for developing self-driven cars which can make your journey more safe and secure.

9. AI in Robotics:

- Artificial Intelligence has a remarkable role in Robotics. Usually, general robots are programmed such that they can perform some repetitive task, but with the help of AI, we can create intelligent robots which can perform tasks with their own experiences without pre-programmed.
- Humanoid Robots are best examples for AI in robotics, recently the intelligent Humanoid robot named as Erica and Sophia has been developed which can talk and behave like humans.

10. AI in Entertainment

- We are currently using some AI based applications in our daily life with some entertainment services such as Netflix or Amazon. With the help of ML/AI algorithms, these services show the recommendations for programs or shows.

11. AI in Agriculture

- Agriculture is an area which requires various resources, labor, money, and time for best result. Now a day's agriculture is becoming digital, and AI is emerging in this field. Agriculture is applying AI as agriculture robotics, solid and crop monitoring, predictive analysis. AI in agriculture can be very helpful for farmers.

12. AI in E-commerce

- AI is providing a competitive edge to the e-commerce industry, and it is becoming more demanding in the e-commerce business. AI is helping shoppers to discover associated products with recommended size, color, or even brand.

13. AI in Education:

- AI can automate grading so that the tutor can have more time to teach. AI chatbot can communicate with students as a teaching assistant.
- AI in the future can be work as a personal virtual tutor for students, which will be accessible easily at any time and any place.

THE FOUNDATIONS OF AI

- **Philosophy:** Logic, methods of reasoning, mind as physical system foundations of learning, language, rationality.
- **Mathematics:** Formal representation and proof algorithms, computation, (un)decidability, (in)tractability, probability.
- **Economics:** utility, decision theory.
- **Neuroscience:** physical substrate for mental activity.
- **Psychology:** phenomena of perception and motor control, experimental techniques.
- **Computer engineering:** building fast computers.
- **Control theory:** design systems that maximize an objective function over time.
- **Linguistics:** knowledge representation, grammar.

HISTORY OF AI

Maturation of Artificial Intelligence (1943-1952)

- **Year 1943:** The first work which is now recognized as AI was done by Warren McCulloch and Walter Pitts in 1943. They proposed a model of **artificial neurons**.
- **Year 1949:** Donald Hebb demonstrated an updating rule for modifying the connection strength between neurons. His rule is now called **Hebbian learning**.
- **Year 1950:** The Alan Turing who was an English mathematician and pioneered Machine learning in 1950. Alan Turing publishes "**Computing Machinery and Intelligence**" in which he proposed a test. The test can check the machine's ability to exhibit intelligent behavior equivalent to human intelligence, called a **Turing test**.

The birth of Artificial Intelligence (1952-1956)

- **Year 1955:** An Allen Newell and Herbert A. Simon created the "first artificial intelligence program" which was named as "**Logic Theorist**".
- **Year 1956:** The word "Artificial Intelligence" first adopted by American Computer scientist John McCarthy at the Dartmouth Conference.

The golden years-Early enthusiasm (1956-1974)

- **Year 1966:** The researchers emphasized developing algorithms which can solve mathematical problems. Joseph Weizenbaum created the first chatbot in 1966, which was named as ELIZA.
- **Year 1972:** The first intelligent humanoid robot was built in Japan which was named as WABOT-1.

The first AI winter (1974-1980)

- AI winter refers to the time period where computer scientist dealt with a severe shortage of funding from government for AI researches.
- During AI winters, an interest of publicity on artificial intelligence was decreased.

A boom of AI (1980-1987)

- **Year 1980:** After AI winter duration, AI came back with "Expert System". Expert systems were programmed that emulate the decision-making ability of a human expert.
- In the Year 1980, the first national conference of the American Association of Artificial Intelligence **was held at Stanford University.**

The second AI winter (1987-1993)

- Again Investors and government stopped in funding for AI research as due to high cost but not efficient result. The expert system such as XCON was very cost effective.

The emergence of intelligent agents (1993-2011)

- **Year 1997:** In the year 1997, IBM Deep Blue beats world chess champion, Gary Kasparov, and became the first computer to beat a world chess champion.
- **Year 2002:** for the first time, AI entered the home in the form of Roomba, a vacuum cleaner.
- **Year 2006:** AI came in the Business world till the year 2006. Companies like Facebook, Twitter, and Netflix also started using AI.

Deep learning, big data and artificial general intelligence (2011-present)

- **Year 2011:** In the year 2011, IBM's Watson won jeopardy, a quiz show, where it had to solve the complex questions as well as riddles. Watson had proved that it could understand natural language and can solve tricky questions quickly.
- **Year 2012:** Google has launched an Android app feature "Google now", which was able to provide information to the user as a prediction.
- **Year 2014:** In the year 2014, Chatbot "Eugene Goostman" won a competition in the infamous "Turing test."
- **Year 2018:** The "Project Debater" from IBM debated on complex topics with two master debaters and also performed extremely well.

AGENTS AND ENVIRONMENTS

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?

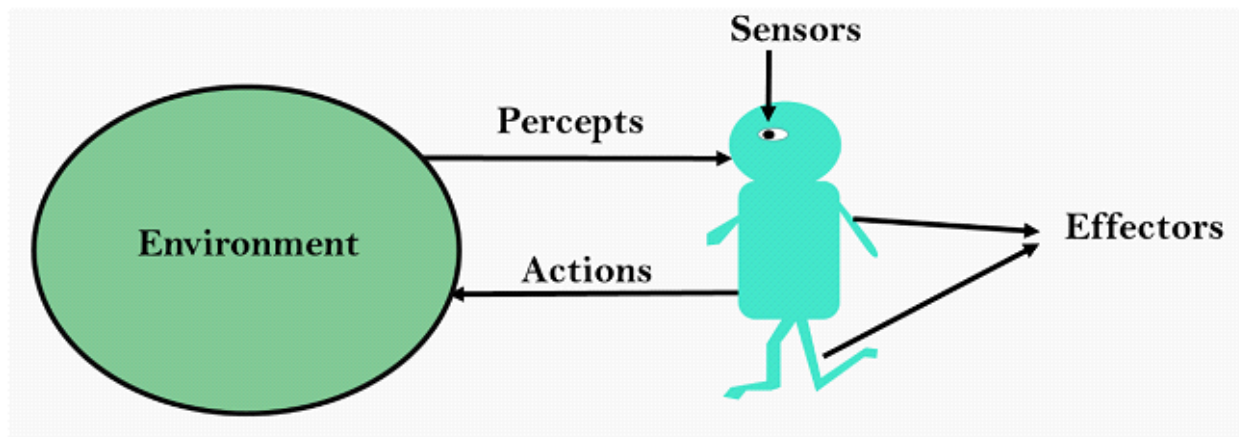
An agent can be anything that perceives its environment through sensors and act upon that environment through actuators. An Agent runs in the cycle of **perceiving, thinking, and acting**. An agent can be:

- **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.

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.



THE CONCEPT OF RATIONALITY

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.

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.

THE NATURE OF ENVIRONMENTS

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.

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.

Features or properties of environment:

As per Russell and Norvig, an environment can have various features from the point of view of an agent:

- Fully observable vs Partially Observable
- Static vs Dynamic
- Discrete vs Continuous
- Deterministic vs Stochastic
- Single-agent vs Multi-agent
- Episodic vs sequential
- Known vs Unknown
- 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 game comes 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.

THE STRUCTURE OF AGENTS

The job of AI is to design an agent program that implements the agent function the mapping from percept to actions. We assume this program will run on some sort of computing device with physical sensors and actuators we call this the architecture.

Agent= architecture + program.

Agent program: The agent program takes the current percept as input from the sensors and returns an action to the actuators. The agent program take just the current percept as input because nothing more is available from the environment; if the agent action need to depend on the entire percept sequence, the agent will have to remember the percept.

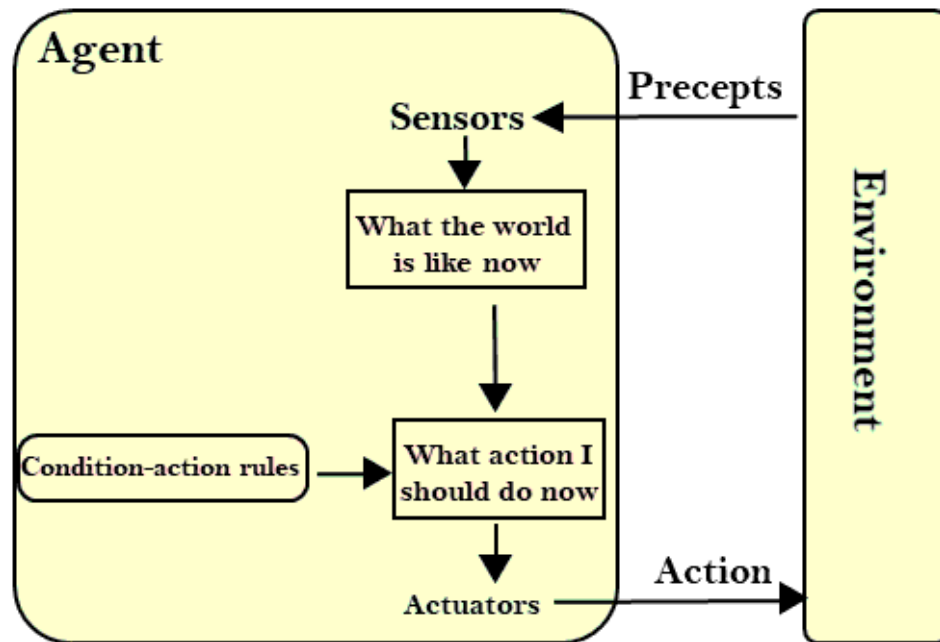
Agents can be grouped into five classes based on their degree of perceived intelligence and capability. All these agents can improve their performance and generate better action over the time. These are given below:

- Simple Reflex Agent
- Model-based reflex agent
- Goal-based agents
- Utility-based agent
- Learning agent

1. Simple Reflex agent:

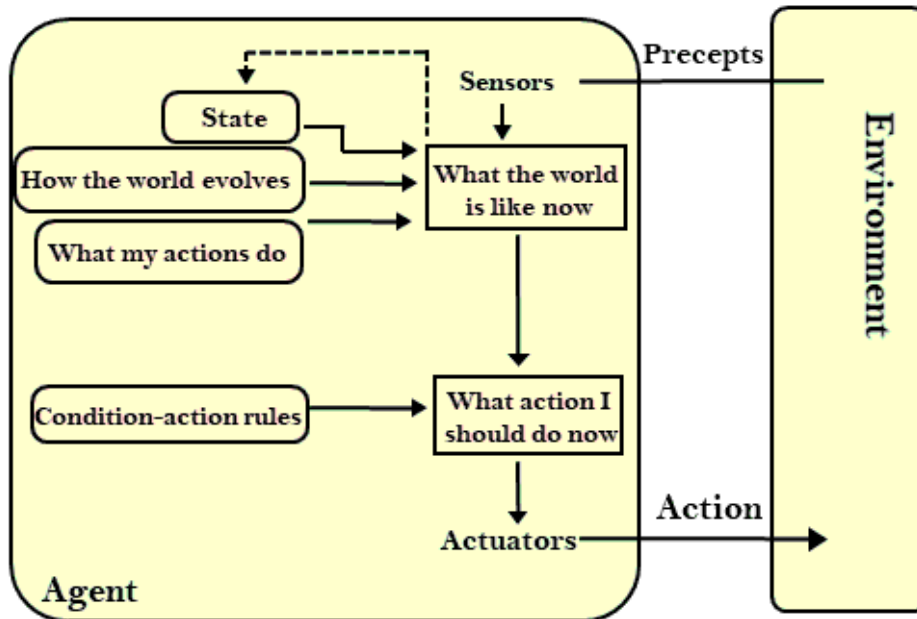
- The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history.
- These agents only succeed in the fully observable environment.
- The Simple reflex agent does not consider any part of percepts history during their decision and action process.
- The Simple reflex agent works on Condition-action rule, which means it maps the current state to action. Such as a Room Cleaner agent, it works only if there is dirt in the room.
- Problems for the simple reflex agent design approach:
 - They have very limited intelligence
 - They do not have knowledge of non-perceptual parts of the current state
 - Mostly too big to generate and to store.

- Not adaptive to changes in the environment.



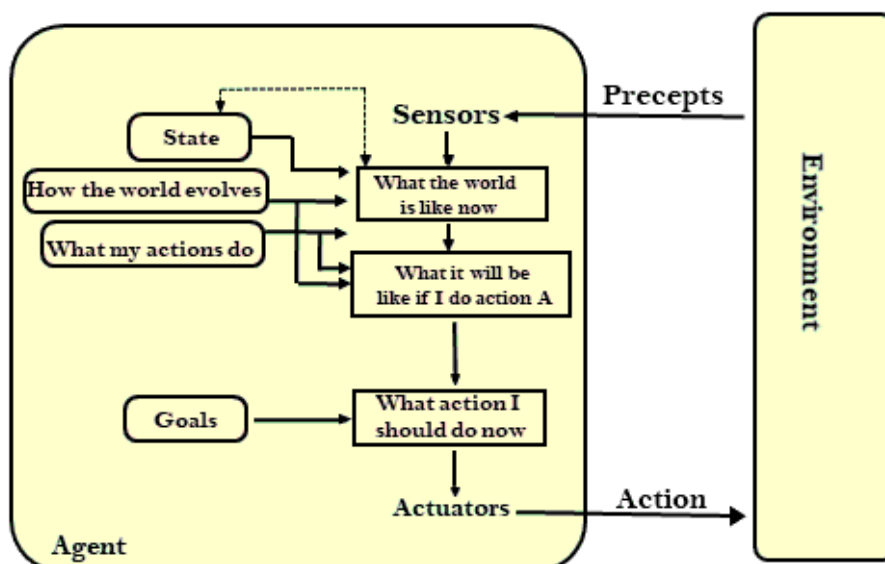
2. Model-based reflex agent:

- The Model-based agent can work in a partially observable environment, and track the situation.
- A model-based agent has two important factors:
 - **Model:** It is knowledge about "how things happen in the world," so it is called a Model-based agent.
 - **Internal State:** It is a representation of the current state based on percept history.
- These agents have the model, "which is knowledge of the world" and based on the model they perform actions.
- Updating the agent state requires information about:
 - a. How the world evolves
 - b. How the agent's action affects the world.



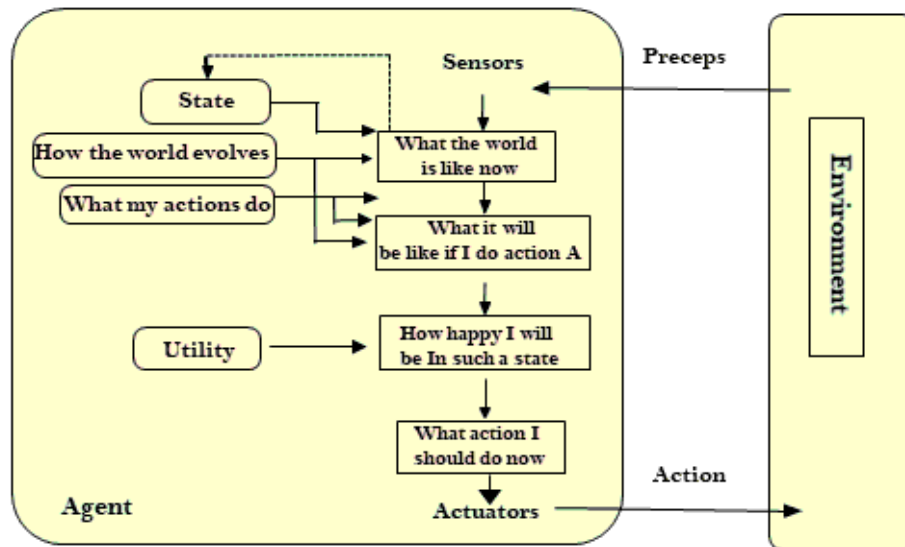
3. Goal-based agents:

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.
- The agent needs to know its goal which describes desirable situations.
- Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.
- They choose an action, so that they can achieve the goal.
- These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not. Such considerations of different scenario are called searching and planning, which makes an agent proactive.



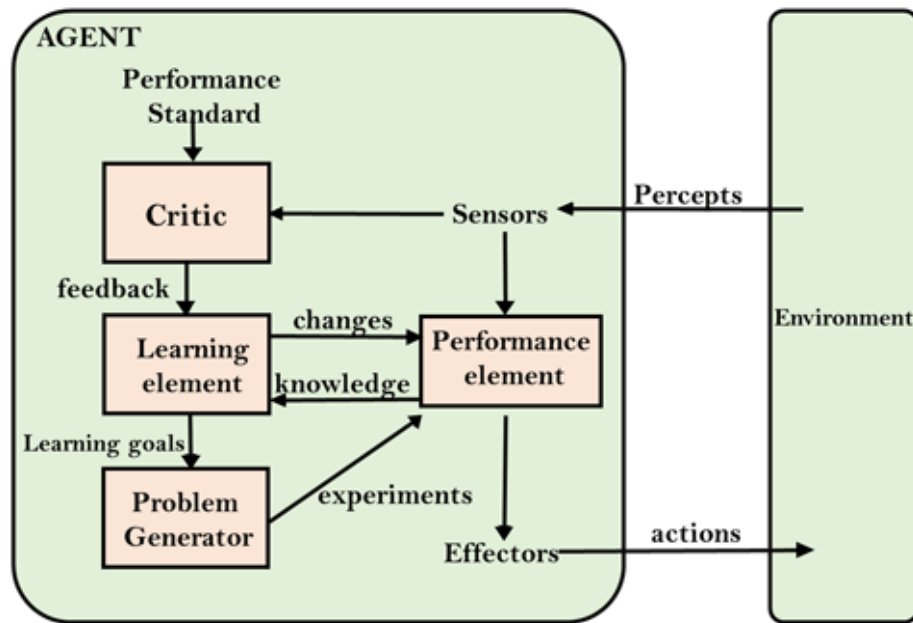
4. Utility-based agents:

- These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Utility-based agent act based not only goals but also the best way to achieve the goal.
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- The utility function maps each state to a real number to check how efficiently each action achieves the goals.



5. Learning Agents:

- A learning agent in AI is the type of agent which can learn from its past experiences, or it has learning capabilities.
- It starts to act with basic knowledge and then able to act and adapt automatically through learning.
- A learning agent has mainly four conceptual components, which are:
 - a. **Learning element:** It is responsible for making improvements by learning from environment
 - b. **Critic:** Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.
 - c. **Performance element:** It is responsible for selecting external action
 - d. **Problem generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.
- 1. Hence, learning agents are able to learn, analyze performance, and look for new ways to improve the performance.



PROBLEM SOLVING AGENTS

In Artificial Intelligence, Search techniques are universal problem-solving methods. **Rational agents** or **Problem-solving agents** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result. Problem-solving agents are the goal-based agents and use atomic representation. In this topic, we will learn various problem-solving search algorithms.

Search Algorithm Terminologies:

- **Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:

A. Search Space: Search space represents a set of possible solutions, which a system may have.

B. Start State: It is a state from where agent begins **the search**.

C. Goal test: It is a function which observe the current state and returns whether the goal state is achieved or not.

1. **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
2. **Actions:** It gives the description of all the available actions to the agent.
3. **Transition model:** A description of what each action do, can be represented as a transition model.
4. **Path Cost:** It is a function which assigns a numeric cost to each path.
5. **Solution:** It is an action sequence which leads from the start node to the goal node.
6. **Optimal Solution:** If a solution has the lowest cost among all solutions

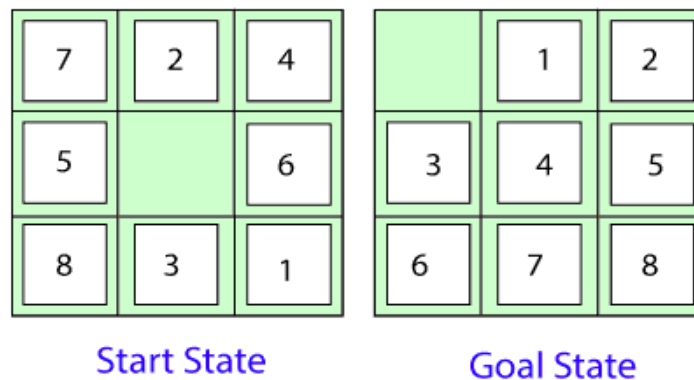
EXAMPLE PROBLEMS

Basically, there are two types of problem approaches:

- **Toy Problem:** It is a concise and exact description of the problem which is used by the researchers to compare the performance of algorithms.
- **Real-world Problem:** It is real-world based problems which require solutions. Unlike a toy problem, it does not depend on descriptions, but we can have a general formulation of the problem.

Some Toy Problems

- **8 Puzzle Problem:** Here, we have a 3×3 matrix with movable tiles numbered from 1 to 8 with a blank space. The tile adjacent to the blank space can slide into that space. The objective is to reach a specified goal state similar to the goal state, as shown in the below figure.
- In the figure, our task is to convert the current state into goal state by sliding digits into the blank space.

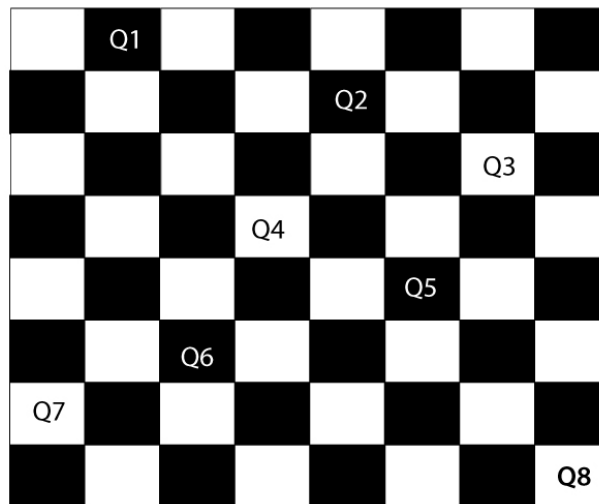


In the above figure, our task is to convert the current(Start) state into goal state by sliding digits into the blank space.

The problem formulation is as follows:

- **States:** It describes the location of each numbered tiles and the blank tile.
- **Initial State:** We can start from any state as the initial state.
- **Actions:** Here, actions of the blank space is defined, i.e., either **left, right, up or down**
- **Transition Model:** It returns the resulting state as per the given state and actions.
- **Goal test:** It identifies whether we have reached the correct goal-state.
- **Path cost:** The path cost is the number of steps in the path where the cost of each step is 1.
- **8-queens problem:** The aim of this problem is to place eight queens on a chessboard in an order where no queen may attack another. A queen can attack other queens either **diagonally or in same row and column.**

From the following figure, we can understand the problem as well as its correct solution.



It is noticed from the above figure that each queen is set into the chessboard in a position where no other queen is placed diagonally, in same row or column. Therefore, it is one right approach to the 8-queens problem.

For this problem, there are two main kinds of formulation:

- **Incremental formulation**: It starts from an empty state where the operator augments a queen at each step.

Following steps are involved in this formulation:

- **States**: Arrangement of any 0 to 8 queens on the chessboard.
- **Initial State**: An empty chessboard
- **Actions**: Add a queen to any empty box.
- **Transition model**: Returns the chessboard with the queen added in a box.
- **Goal test**: Checks whether 8-queens are placed on the chessboard without any attack.
- **Path cost**: There is no need for path cost because only final states are counted.

In this formulation, there is approximately 1.8×10^{14} possible sequence to investigate.

- **Complete-state formulation**: It starts with all the 8-queens on the chessboard and moves them around, saving from the attacks.

Following steps are involved in this formulation:

- **States**: Arrangement of all the 8 queens' one per column with no queen attacking the other queen.
- **Actions**: Move the queen at the location where it is safe from the attacks.

This formulation is better than the incremental formulation as it reduces the state space from 1.8×10^{14} to 2057, and it is easy to find the solutions.

Some Real-world problems

- **Travelling salesperson problem (TSP)**: It is a **touring problem** where the salesman can visit each city only once. The objective is to find the shortest tour and sell-out the stuff in each city.

SEARCHING FOR SOLUTIONS

We have seen many problems. Now, there is a need to search for solutions to solve them.

In this section, we will understand how searching can be used by the agent to solve a problem.

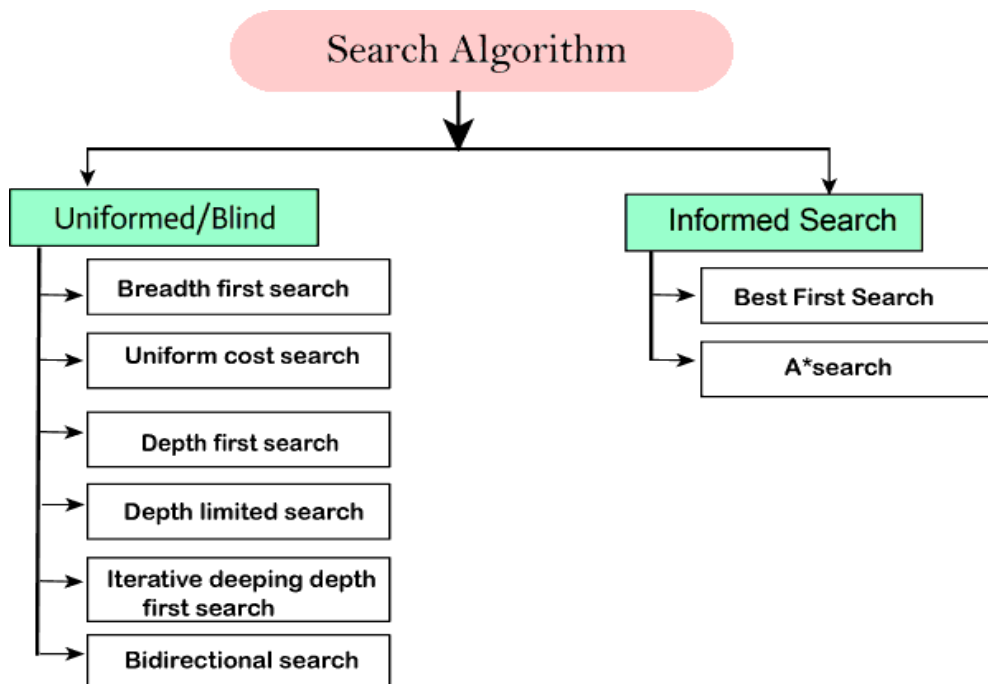
For solving different kinds of problem, an agent makes use of different strategies to reach the goal by searching the best possible algorithms. This process of searching is known as **search strategy**.

Properties of Search Algorithms:

1. Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:
 - **Completeness:** A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.
 - **Optimality:** If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.
 - **Time Complexity:** Time complexity is a measure of time for an algorithm to complete its task.
 - **Space Complexity:** It is the maximum storage space required at any point during the search, as the complexity of the problem.

Types of search algorithms:

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.



UNINFORMED SEARCH STRATEGIES

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

1. Breadth-first Search:

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

Advantages:

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

Disadvantages:

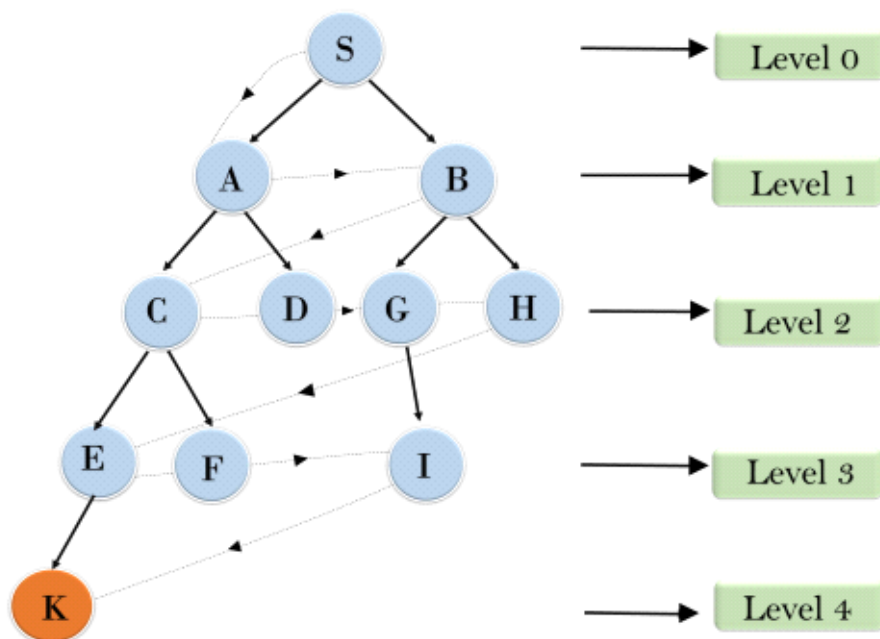
- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example:

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

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

Breadth First Search



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

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

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

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

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

2. Depth-first Search:

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

Advantages:

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

Disadvantages:

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

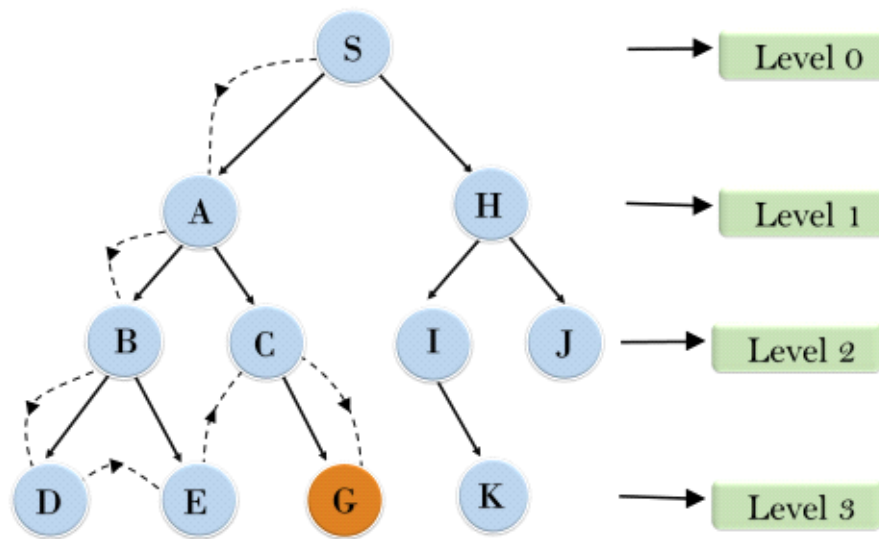
Example:

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

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

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

Depth First Search



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

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

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

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

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

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

3. Depth-Limited Search Algorithm:

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that problem does not have any solution.
- Cut-off failure value: It defines no solution for the problem within a given depth limit.

Advantages:

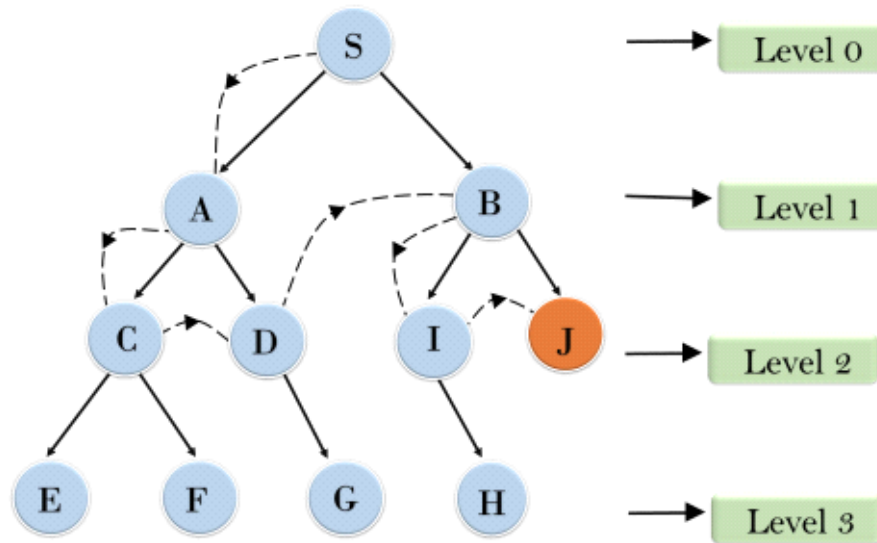
Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Example:

Depth Limited Search



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is $O(b^l)$.

Space Complexity: Space complexity of DLS algorithm is $O(b \times l)$.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $l > d$.

Informed Search Algorithms

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

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

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

Admissibility of the heuristic function is given as:

- $h(n) \leq h^*(n)$

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

Pure Heuristic Search:

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

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

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

- **Best First Search Algorithm(Greedy search)**
- **A* Search Algorithm**

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

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

- $f(n) = g(n)$.

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

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

Best first search algorithm:

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n , from the OPEN list which has the lowest value of $h(n)$, and places it in the CLOSED list.
- **Step 4:** Expand the node n , and generate the successors of node n .
- **Step 5:** Check each successor of node n , and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function $f(n)$, and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

Advantages:

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

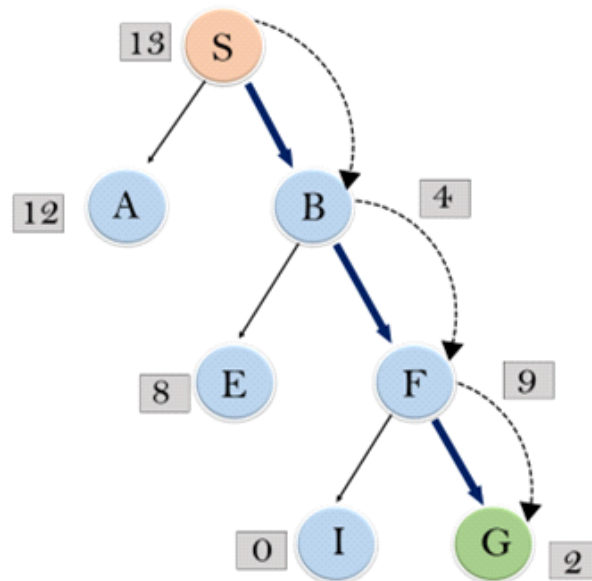
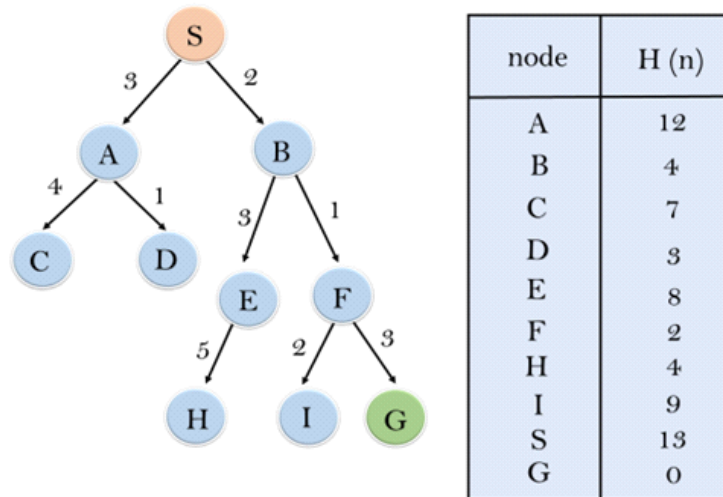
Disadvantages:

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

Example:

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

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



Expand the nodes of S and put in the CLOSED list

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

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

Iteration2: Open[E,F,A],Closed[S,B]
: Open [E, A], Closed [S, B, F]

Iteration3: Open[I,G,E,A],Closed[S,B,F]
: Open [I, E, A], Closed [S, B, F, G]

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

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

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

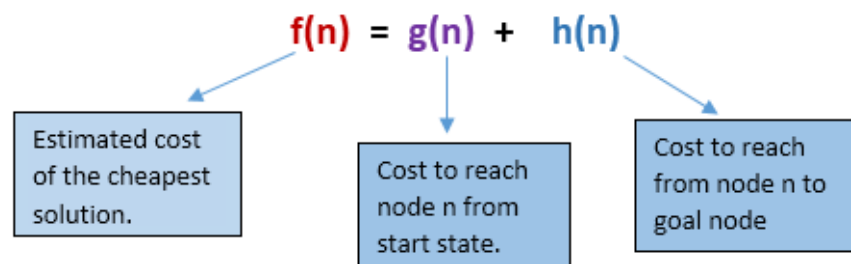
Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

2.) A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function $h(n)$, and cost to reach the node n from the start state $g(n)$. It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses $g(n)+h(n)$ instead of $g(n)$.

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



Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

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

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

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

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

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

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

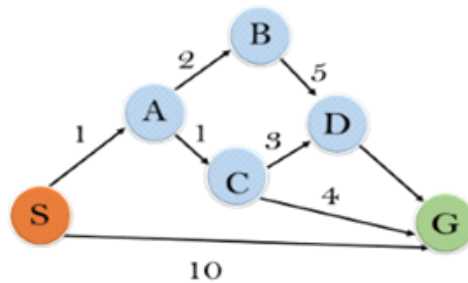
Disadvantages:

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

Example:

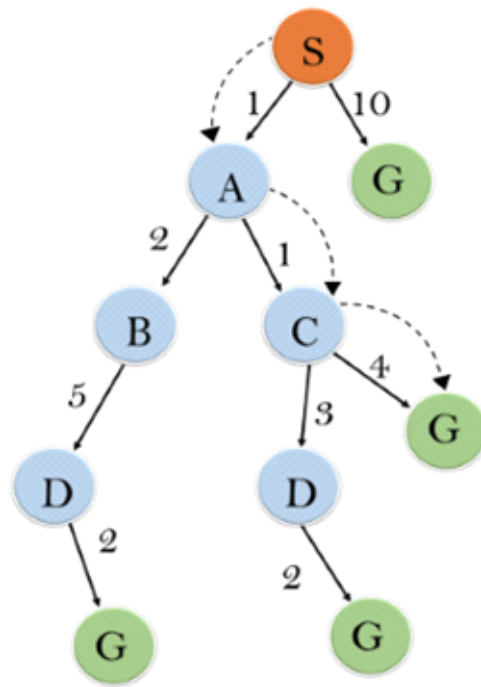
In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the $f(n)$ of each state using the formula $f(n) = g(n) + h(n)$, where $g(n)$ is the cost to reach any node from start state.

Here we will use OPEN and CLOSED list.



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

Solution:



Initialization: {(S, 5)}

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

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

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

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

Complete: A* algorithm is complete as long as:

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

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

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

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

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

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

D.N.R.COLLEGE(AUTONOMOUS):BHIMAVARAM
M.C.A DEPARTMENT



ARTIFICIAL INTELLIGENCE

Presented by
L.SOWJANYA

UNIT II

Heuristic Functions, Local-Search Algorithms and Optimization Problems: Hill Climbing, Simulated Annealing, Genetic Algorithms; Constraint Satisfaction Problems, Backtracking Search For CSPs, Games, Optimal Decisions in Games

Knowledge Based Agents, The Wumpus World, Logic, Propositional Logic, Reasoning

Patterns in Propositional Logic, Syntax and Semantics of First Order Logic, Using First Order Logic, Inference in First-Order Logic: Unification, Resolution.

UNIT- 2

Local Search Algorithms and Optimization Problem

The informed and uninformed search expands the nodes systematically in two ways:

- keeping different paths in the memory and
- selecting the best suitable path,

Which leads to a solution state required to reach the goal node? But beyond these “classical search algorithms,” we have some “local search algorithms” where the path cost does not matter, and only focus on solution-state needed to reach the goal node.

A local search algorithm completes its task by traversing on a single current node rather than multiple paths and following the neighbours of that node generally.

Although local search algorithms are not systematic, still they have the following two advantages:

- Local search algorithms use a very little or constant amount of memory as they operate only on a single path.
- Most often, they find a reasonable solution in large or infinite state spaces where the classical or systematic algorithms do not work.

The local search algorithm works for pure optimized problems. A pure optimization problem is one where all the nodes can give a solution. But the target is to find the best state out of all according to the **objective function**. Unfortunately, the pure optimization problem fails to find high-quality solutions to reach the goal state from the current state.

Note: An objective function is a function whose value is either minimized or maximized in different contexts of the optimization problems. In the case of search algorithms, an objective function can be the path cost for reaching the goal node, etc.

Hill Climbing Algorithm

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.

- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

Features of Hill Climbing:

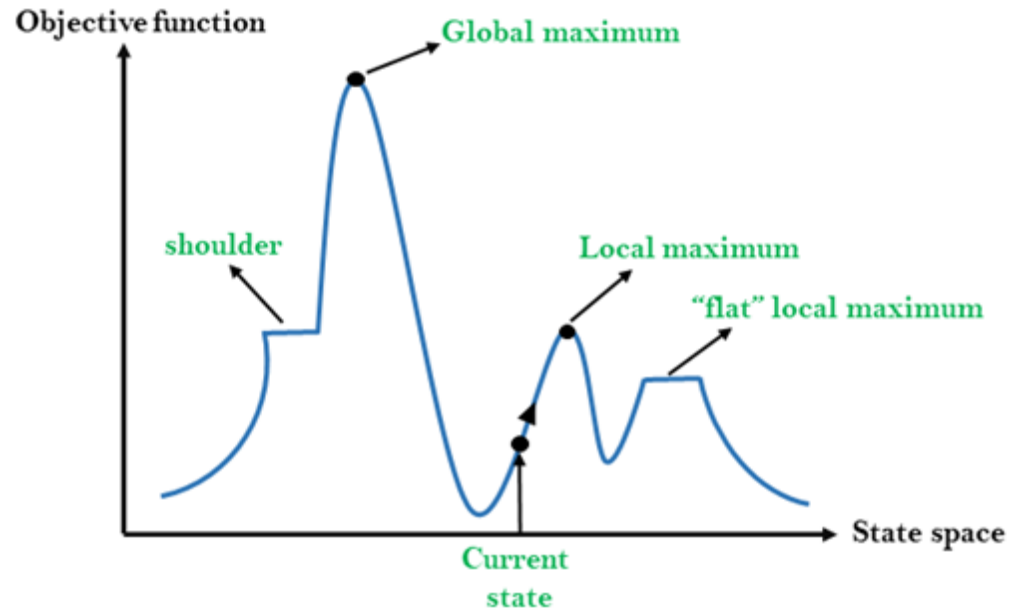
Following are some main features of Hill Climbing Algorithm:

- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing:

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

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



Different regions in the state space landscape:

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

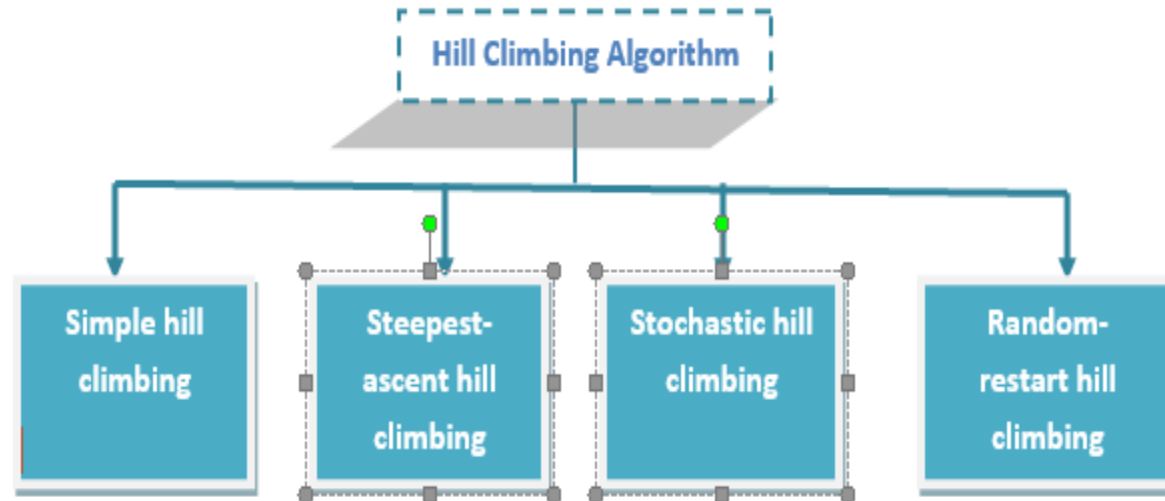
Global Maximum: Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

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

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

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

Types of Hill Climbing Algorithm:



1. Simple Hill Climbing:

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

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

Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.

- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
 - a. If it is goal state, then return success and quit.
 - b. Else if it is better than the current state then assign new state as a current state.
 - c. Else if not better than the current state, then return to step2.

Step 5: Exit.

2. Steepest-Ascent hill climbing:

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

Algorithm for Steepest-Ascent hill climbing:

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

Step 3: Exit.

3. Stochastic hill climbing:

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

4.Random-restart hill climbing

Random-restart algorithm is based on **try and try strategy**. It iteratively searches the node and selects the best one at each step until the goal is not found. The success depends most commonly on the shape of the hill. If there are few plateaus, local maxima, and ridges, it becomes easy to reach the destination.

Simulated Annealing:

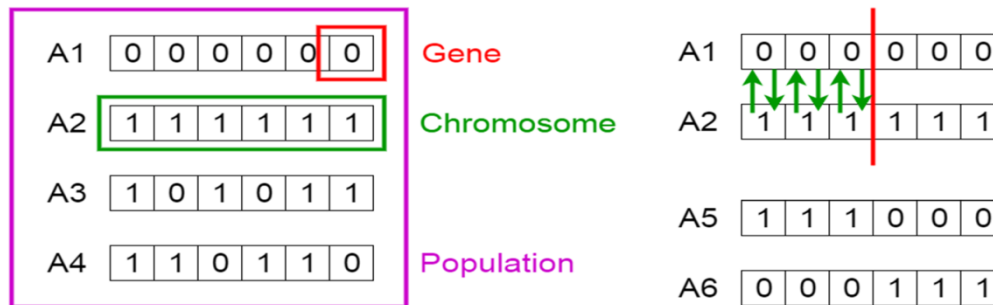
A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum. And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient. **Simulated Annealing** is an algorithm which yields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

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

Genetic algorithms

A **genetic algorithm** is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Genetic Algorithms



Notion of Natural Selection:

The process of natural selection starts with the selection of fittest individuals from a population. They produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

This notion can be applied for a search problem. We consider a set of solutions for a problem and select the set of best ones out of them.

Five phases are considered in a genetic algorithm.

1. Initial population

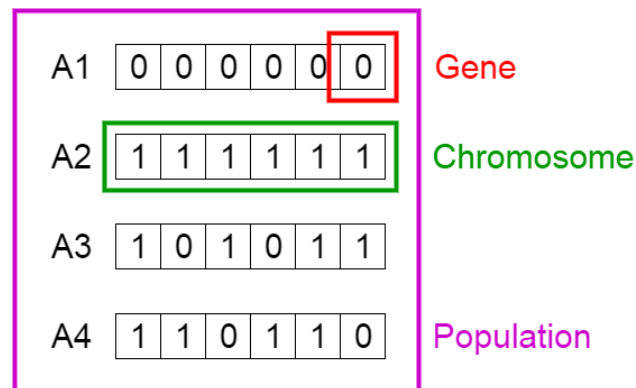
2. Fitness function
3. Selection
4. Crossover
5. Mutation

Initial Population

The process begins with a set of individuals which is called a **Population**. Each individual is a solution to the problem you want to solve.

An individual is characterized by a set of parameters (variables) known as **Genes**. Genes are joined into a string to form a **Chromosome** (solution).

In a genetic algorithm, the set of genes of an individual is represented using a string, in terms of an alphabet. Usually, binary values are used (string of 1s and 0s). We say that we encode the genes in a chromosome.



Population, Chromosomes and Genes

Fitness Function

The **fitness function** determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a **fitness score** to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

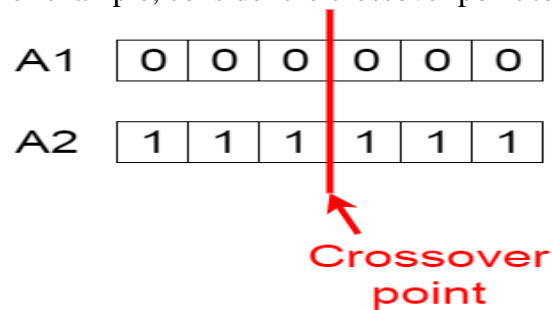
Selection

The idea of **selection** phase is to select the fittest individuals and let them pass their genes to the next generation. Two pairs of individuals (**parents**) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

Crossover

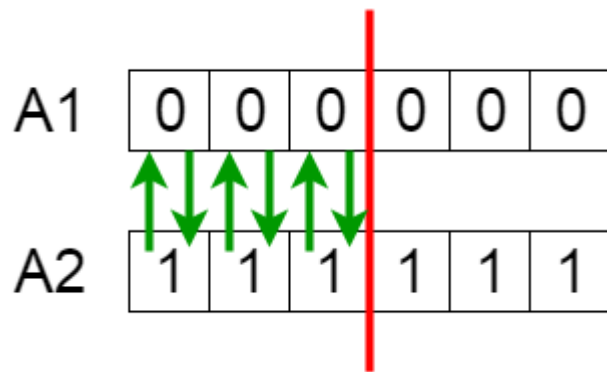
Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a **crossover point** is chosen at random from within the genes.

For example, consider the crossover point to be 3 as shown below.



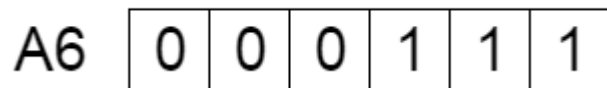
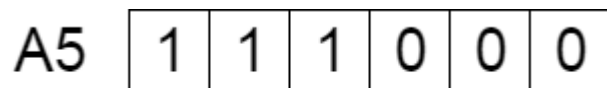
Crossover point

Offspring are created by exchanging the genes of parents among themselves until the crossover point is reached.



Exchanging genes among parents

The new offspring are added to the population.



New offspring

Mutation

In certain new offspring formed, some of their genes can be subjected to a **mutation** with a low random probability. This implies that some of the bits in the bit string can be flipped.

Before Mutation

A5

1	1	1	0	0	0
---	---	---	---	---	---

After Mutation

A5

1	1	0	1	1	0
---	---	---	---	---	---

Mutation: Before and After

Mutation occurs to maintain diversity within the population and prevent premature convergence.

Termination

The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation). Then it is said that the genetic algorithm has provided a set of solutions to our problem.

Pseudocode

START

Generate the initial population

Compute fitness

REPEAT

Selection

Crossover

Mutation

Compute fitness

UNTIL population has converged

STOP

Constraint satisfaction Problems

Definition

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

- A finite set of **variables** which stores the solution. ($V = \{V_1, V_2, V_3, \dots, V_n\}$)
- A set of **discrete** values known as **domain** from which the solution is picked. ($D = \{D_1, D_2, D_3, \dots, D_n\}$)
- A finite set of **constraints**. ($C = \{C_1, C_2, C_3, \dots, C_n\}$)

Please note that the elements in the domain can be both continuous and discrete but in AI, we generally only deal with discrete values.

Also, note that all these sets should be finite except for the domain set. Each variable in the variable set can have different domains. For example, consider the Sudoku problem again. Suppose that a row, column and block already have 3,5 and 7 filled in. Then the domain for all the variables in that row, column and block will be $\{1,2,4,6,8,9\}$.

Popular Problems of CSP

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

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

Converting problems to CSPs

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

- **Step 1:** Create a variable set.
- **Step 2:** Create a domain set.

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

Example:

Cryptarithmic Problem

Cryptarithmic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement either with alphabets or other symbols. In cryptarithmic problem, the digits (0-9) get substituted by some possible alphabets or symbols. The task in cryptarithmic problem is to substitute each digit with an alphabet to get the result arithmetically correct.

We can perform all the arithmetic operations on a given cryptarithmic problem.

The rules or constraints on a cryptarithmic problem are as follows:

- There should be a unique digit to be replaced with a unique alphabet.
- The result should satisfy the predefined arithmetic rules, i.e., $2+2=4$, nothing else.
- Digits should be from **0-9** only.
- There should be only one carry forward, while performing the addition operation on a problem.
- The problem can be solved from both sides, i.e., **lefthand side (L.H.S), or righthand side (R.H.S)**

Let's understand the cryptarithmic problem as well its constraints better with the help of an example:

- Given a cryptarithmic problem, i.e., **S E N D + M O R E = M O N E Y**

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

In this example, add both terms **S E N D** and **M O R E** to bring **M O N E Y** as a result.

Follow the below steps to understand the given problem by breaking it into its subparts:

- Starting from the left hand side (L.H.S) , the terms are **S** and **M**. Assign a digit which could give a satisfactory result. Let's assign **S->9** and **M->1**.

$$\begin{array}{r}
 \mathbf{S} \\
 + \mathbf{M} \\
 \hline
 \mathbf{M O}
 \end{array}
 \longrightarrow
 \begin{array}{r}
 \mathbf{9} \\
 + \mathbf{1} \\
 \hline
 \mathbf{10}
 \end{array}$$

Hence, we get a satisfactory result by adding up the terms and got an assignment for **O** as **O->0** as well.

- Now, move ahead to the next terms **E** and **O** to get **N** as its output.

$$\begin{array}{r}
 \mathbf{E} \\
 + \mathbf{O} \\
 \hline
 \mathbf{N}
 \end{array}
 \xrightarrow{\text{X}}
 \begin{array}{r}
 \mathbf{5} \\
 + \mathbf{0} \\
 \hline
 \mathbf{5}
 \end{array}$$

Adding E and O, which means 5+0=0, which is not possible because according to cryptarithmic constraints, we cannot assign the same digit to two letters. So, we need to think more and assign some other value.

$$\begin{array}{r}
 \mathbf{E} \\
 + \mathbf{O} \\
 \hline
 \mathbf{N}
 \end{array}
 \longrightarrow
 \begin{array}{r}
 \textcircled{1} \longleftarrow \text{carry} \\
 \mathbf{5} \\
 + \mathbf{0} \\
 \hline
 \mathbf{6}
 \end{array}$$

Note: When we will solve further, we will get one carry, so after applying it, the answer will be satisfied.

- Further, adding the next two terms **N** and **R** we get,

$$\begin{array}{r}
 N \\
 + R \\
 \hline
 E \\
 \hline
 \end{array}
 \xrightarrow{\text{X}}
 \begin{array}{r}
 6 \\
 + 8 \\
 \hline
 14 \\
 \hline
 \end{array}$$

But, we have already assigned $E \rightarrow 5$. Thus, the above result does not satisfy the values because we are getting a different value for E . So, we need to think more.

Again, after solving the whole problem, we will get a carryover on this term, so our answer will be satisfied.

$$\begin{array}{r}
 N \\
 + R \\
 \hline
 E \\
 \hline
 \end{array}
 \xrightarrow{\text{1}}
 \begin{array}{r}
 \textcircled{1} \\
 6 \\
 + 8 \\
 \hline
 15 \\
 \hline
 \end{array}$$

← carry

where 1 will be carry forward to the above term

Let's move ahead.

- Again, on adding the last two terms, i.e., the rightmost terms D and E , we get Y as its result.

$$\begin{array}{r}
 D \\
 + E \\
 \hline
 Y \\
 \hline
 \end{array}
 \xrightarrow{\text{1}}
 \begin{array}{r}
 7 \\
 + 5 \\
 \hline
 12 \\
 \hline
 \end{array}$$

where 1 will be carry forward to the above term

- Keeping all the constraints in mind, the final resultant is as follows:

$$\begin{array}{r} \text{SEND} \\ + \text{MORE} \\ \hline \text{MONEY} \end{array}$$

- Below is the representation of the assignment of the digits to the alphabets.

S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2

Back tracking search for csp's:

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

The algorithm above actually has a lot in common with the permutations algorithm, it pretty much just creates all arrangements of the mapping from characters to digits and tries each until one works or all have been successfully tried. For a large puzzle, this could take a while.

A smarter algorithm could take into account the structure of the puzzle and avoid going down dead-end paths. For example, if we assign the characters starting from the one's place and moving to the left, at each stage, we can verify the correctness of what we have so far before we continue onwards. This definitely complicates the code but leads to a tremendous improvement in efficiency, making it much more feasible to solve large puzzles.

Below pseudocode, in this case, has more special cases, but the same general design

- Start by examining the rightmost digit of the topmost row, with a carry of 0
- If we are beyond the leftmost digit of the puzzle, return true if no carry, false otherwise
- If we are currently trying to assign a char in one of the addends
 - If char already assigned, just recur on the row beneath this one, adding value into the sum
 - If not assigned, then
 - for (every possible choice among the digits not in use)
 - make that choice and then on row beneath this one, if successful, return true
 - if !successful, unmake assignment and try another digit
 - return false if no assignment worked to trigger backtracking
- Else if trying to assign a char in the sum
- If char assigned & matches correct,
 - recur on next column to the left with carry, if success return true,
- If char assigned & doesn't match, return false
- If char unassigned & correct digit already used, return false
- If char unassigned & correct digit unused,
 - assign it and recur on next column to left with carry, if success return true
- return false to trigger backtracking

Games:

Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

Types of Games :

	Deterministic	Chance Moves
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

- **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.
- **Imperfect information:** If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called as stochastic games.
Example: Backgammon, Monopoly, Poker, etc.

Formalization of the problem:

A game can be defined as a type of search in AI which can be formalized of the following elements:

- **Initial state:** It specifies how the game is set up at the start.
- **Player(s):** It specifies which player has moved in the state space.
- **Action(s):** It returns the set of legal moves in state space.
- **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.
- **Terminal-Test(s):** Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.
- **Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p . It is also called payoff function. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, $\frac{1}{2}$. And for tic-tac-toe, utility values are +1, -1, and 0.

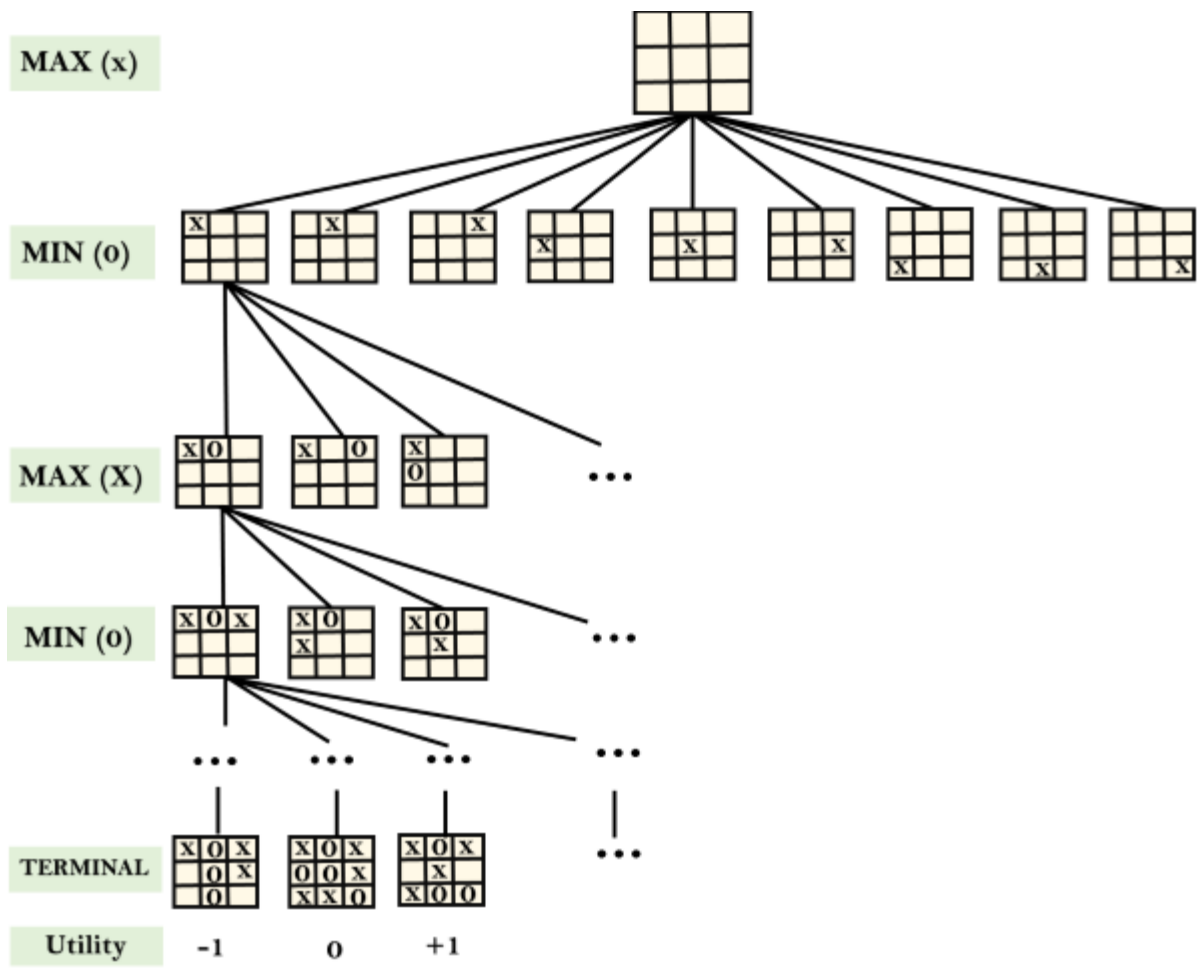
Game tree:

A game tree is a tree where nodes of the tree are the game states and Edges of the tree are the moves by players. Game tree involves initial state, actions function, and result Function.

Example: Tic-Tac-Toe game tree:

The following figure is showing part of the game-tree for tic-tac-toe game. Following are some key points of the game:

- There are two players MAX and MIN.
- Players have an alternate turn and start with MAX.
- MAX maximizes the result of the game tree
- MIN minimizes the result.



Example Explanation:

- From the initial state, MAX has 9 possible moves as he starts first. MAX place x and MIN place o, and both player plays alternatively until we reach a leaf node where one player has three in a row or all squares are filled.
- Both players will compute each node, minimax, the minimax value which is the best achievable utility against an optimal adversary.

- Suppose both the players are well aware of the tic-tac-toe and playing the best play. Each player is doing his best to prevent another one from winning. MIN is acting against Max in the game.
- So in the game tree, we have a layer of Max, a layer of MIN, and each layer is called as **Ply**. Max place x, then MIN puts o to prevent Max from winning, and this game continues until the terminal node.
- In this either MIN wins, MAX wins, or it's a draw. This game-tree is the whole search space of possibilities that MIN and MAX are playing tic-tac-toe and taking turns alternately.

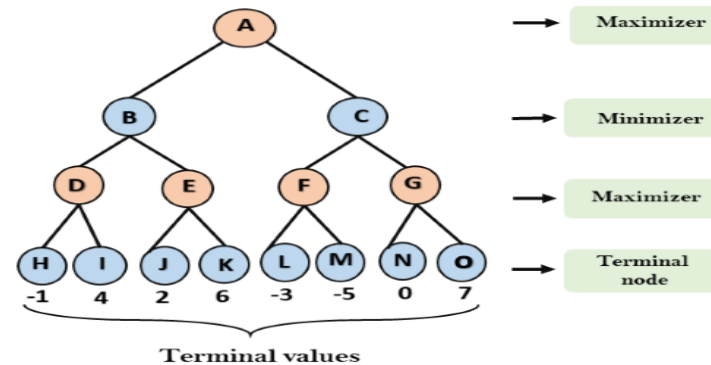
In a given game tree, the optimal strategy can be determined from the minimax value of each node, which can be written as MINIMAX(n). MAX prefer to move to a state of maximum value and MIN prefer to move to a state of minimum value then:

$$\text{For a state } S \text{ MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{If } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{If } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{If } \text{PLAYER}(s) = \text{MIN}. \end{cases}$$

Mini-Max Algorithm

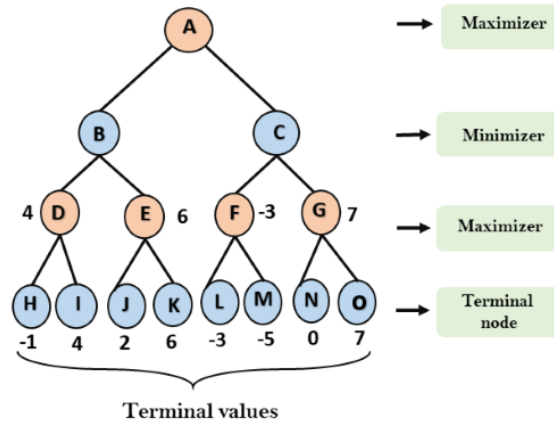
- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly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

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



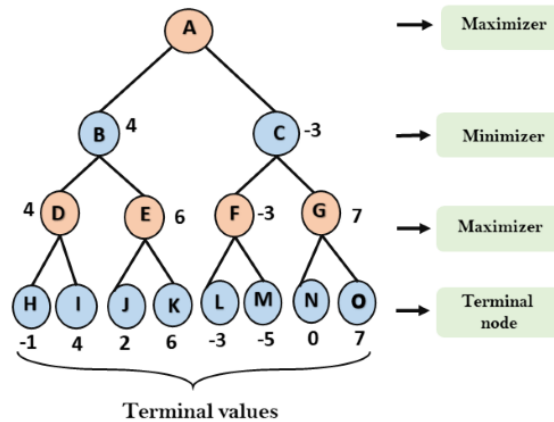
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is $-\infty$, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- For node D $\max(-1, -\infty) \Rightarrow \max(-1, 4) = 4$
- For Node E $\max(2, -\infty) \Rightarrow \max(2, 6) = 6$
- For Node F $\max(-3, -\infty) \Rightarrow \max(-3, -5) = -3$
- For node G $\max(0, -\infty) = \max(0, 7) = 7$



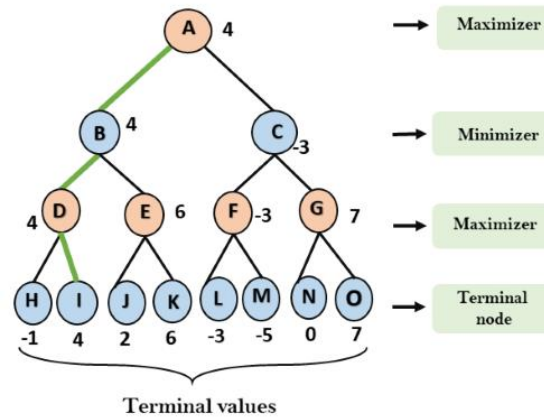
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with $+\infty$, and will find the 3rd layer node values.

- For node B = $\min(4, 6) = 4$
- For node C = $\min(-3, 7) = -3$



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

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



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

Properties of Mini-Max algorithm:

- **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is $O(b^m)$, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is $O(bm)$.

Knowledge-Based Agent:

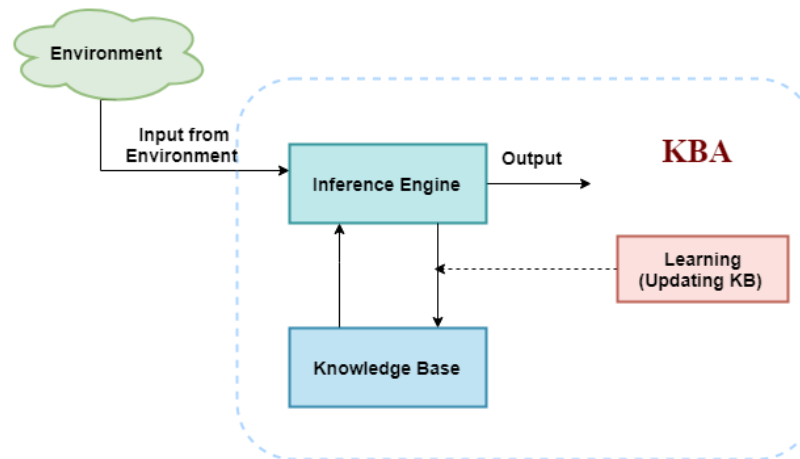
- An intelligent agent needs **knowledge** about the real world for taking decisions and **reasoning** to act efficiently.

- Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently.**
- Knowledge-based agents are composed of two main parts:
 - **Knowledge-base and**
 - **Inference system.**

A knowledge-based agent must able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent Should be able to incorporate new percepts
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world

The architecture of knowledge-based agent:



The above diagram is representing a generalized architecture for a knowledge-based agent. The knowledge-based agent (KBA) take input from the environment by perceiving the environment. The input is taken by the inference engine of the agent and which also

communicate with KB to decide as per the knowledge store in KB. The learning element of KBA regularly updates the KB by learning new knowledge.

Knowledge base: Knowledge-base is a central component of a knowledge-based agent, it is also known as KB. It is a collection of sentences (here 'sentence' is a technical term and it is not identical to sentence in English). These sentences are expressed in a language which is called a knowledge representation language. The Knowledge-base of KBA stores fact about the world.

Use of knowledge base:

Knowledge-base is required for updating knowledge for an agent to learn with experiences and take action as per the knowledge.

Inference system:

Inference means deriving new sentences from old. Inference system allows us to add a new sentence to the knowledge base. A sentence is a proposition about the world. Inference system applies logical rules to the KB to deduce new information.

Inference system generates new facts so that an agent can update the KB. An inference system works mainly in two rules which are given as:

- **Forward chaining**
- **Backward chaining**

Operations Performed by KBA

Following are three operations which are performed by KBA in order to show the intelligent behavior:

1. **TELL:** This operation tells the knowledge base what it perceives from the environment.
2. **ASK:** This operation asks the knowledge base what action it should perform.
3. **Perform:** It performs the selected action.

The knowledge-based agent takes percept as input and returns an action as output. The agent maintains the knowledge base, KB, and it initially has some background knowledge of the real world. It also has a counter to indicate the time for the whole process, and this counter is initialized with zero.

Each time when the function is called, it performs its three operations:

- Firstly it TELLS the KB what it perceives.
- Secondly, it asks KB what action it should take
- Third agent program TELLS the KB that which action was chosen.

The MAKE-PERCEPT-SENTENCE generates a sentence as setting that the agent perceived the given percept at the given time.

The MAKE-ACTION-QUERY generates a sentence to ask which action should be done at the current time.
















MAKE-ACTION-SENTENCE generates a sentence which asserts that the chosen action was executed.

Wumpus world:

The Wumpus world is a simple world example to illustrate the worth of a knowledge-based agent and to represent knowledge representation. It was inspired by a video game **Hunt the Wumpus** by Gregory Yob in 1973.

The Wumpus world is a cave which has 4/4 rooms connected with passageways. So there are total 16 rooms which are connected with each other. We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room. The Wumpus can be shot by the agent, but the agent has a single arrow. In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever. The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold. So the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus. The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.

Following is a sample diagram for representing the Wumpus world. It is showing some rooms with Pits, one room with Wumpus and one agent at (1, 1) square location of the world.

4	 Stench		 Breeze	 PIT
3	 Wumpus	 Breeze  Stench  Gold	 PIT	 Breeze
2	 Stench		 Breeze	
1	 Agent	 Breeze	 PIT	 Breeze
	1	2	3	4

There are also some components which can help the agent to navigate the cave. These components are given as follows:

- The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.
- The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
- There will be glitter in the room if and only if the room has gold.
- The Wumpus can be killed by the agent if the agent is facing to it, and Wumpus will emit a horrible scream which can be heard anywhere in the cave.

PEAS description of Wumpus world:

To explain the Wumpus world we have given PEAS description as below:

Performance measure:

- +1000 reward points if the agent comes out of the cave with the gold.
- -1000 points penalty for being eaten by the Wumpus or falling into the pit.
- -1 for each action, and -10 for using an arrow.
- The game ends if either agent dies or came out of the cave.

Environment:

- A 4*4 grid of rooms.
- The agent initially in room square [1, 1], facing toward the right.
- Location of Wumpus and gold are chosen randomly except the first square [1,1].
- Each square of the cave can be a pit with probability 0.2 except the first square.

Actuators:

- Left turn,
- Right turn
- Move forward
- Grab
- Release
- Shoot.

Sensors:

- The agent will perceive the **stench** if he is in the room adjacent to the Wumpus. (Not diagonally).
- The agent will perceive **breeze** if he is in the room directly adjacent to the Pit.
- The agent will perceive the **glitter** in the room where the gold is present.
- The agent will perceive the **bump** if he walks into a wall.
- When the Wumpus is shot, it emits a horrible **scream** which can be perceived anywhere in the cave.
- These percepts can be represented as five element list, in which we will have different indicators for each sensor.
- Example if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as:
[Stench, Breeze, None, None, None].

The Wumpus world Properties:

- **Partially observable:** The Wumpus world is partially observable because the agent can only perceive the close environment such as an adjacent room.
- **Deterministic:** It is deterministic, as the result and outcome of the world are already known.
- **Sequential:** The order is important, so it is sequential.
- **Static:** It is static as Wumpus and Pits are not moving.
- **Discrete:** The environment is discrete.
- **One agent:** The environment is a single agent as we have one agent only and Wumpus is not considered as an agent.

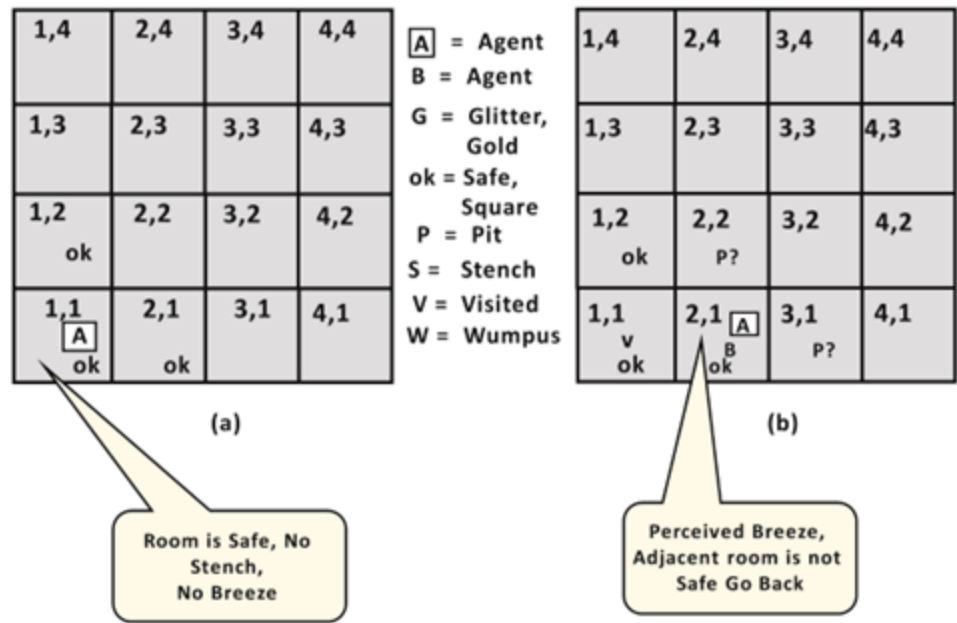
Exploring the Wumpus world:

Now we will explore the Wumpus world and will determine how the agent will find its goal by applying logical reasoning.

Agent's First step:

Initially, the agent is in the first room or on the square [1,1], and we already know that this room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add symbol OK. Symbol A is used to represent agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, W for Wumpus.

At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.



Agent's second Step:

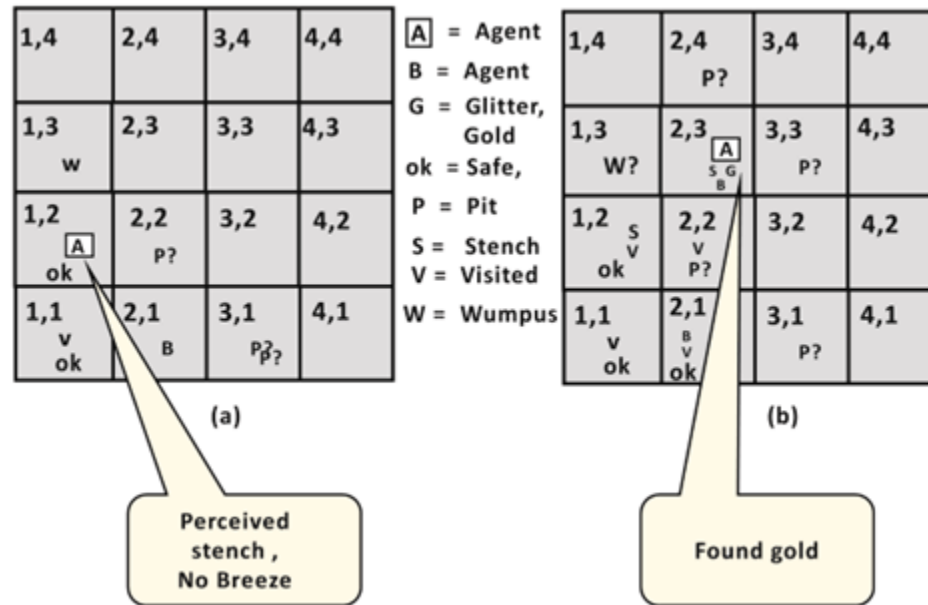
Now agent needs to move forward, so it will either move to [1, 2], or [2,1]. Let's suppose agent moves to the room [2, 1], at this room agent perceives some breeze which means Pit is around this room. The pit can be in [3, 1], or [2,2], so we will add symbol P? to say that, is this Pit room?

Now agent will stop and think and will not make any harmful move. The agent will go back to the [1, 1] room. The room [1,1], and [2,1] are visited by the agent, so we will use symbol V to represent the visited squares.

Agent's third step:

At the third step, now agent will move to the room [1,2] which is OK. In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby. But Wumpus cannot be in the room [1,1] as by rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]). Therefore agent infers that Wumpus is in the room [1,3], and in current state, there is no

breeze which means in [2,2] there is no Pit and no Wumpus. So it is safe, and we will mark it OK, and the agent moves further in [2,2].



Agent's fourth step:

At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3]. At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

Logic:

Logic, as per the definition of the Oxford dictionary, is "the reasoning conducted or assessed according to strict principles and validity". In [Artificial Intelligence](#) also, it carries somewhat the same meaning. Logic can be defined as the proof or validation behind

any reason provided. It is simply the ‘dialectics behind reasoning’. It was important to include logic in Artificial Intelligence because we want our [agent](#) (system) to think and act humanly, and for doing so, it should be capable of taking any decision based on the current situation. If we talk about normal human behavior, then a decision is made by choosing an option from the various available options. There are reasons behind selecting or rejecting an option. So, our artificial agent should also work in this manner.

While taking any decision, the agent must provide specific reasons based on which the decision was taken. And this reasoning can be done by the agent only if the agent has the capability of understanding the logic.

Types of logics in Artificial Intelligence:

In artificial Intelligence, we deal with two **types of logics**:

1. Deductive logic
2. Inductive logic

1) Deductive logic

In deductive logic, the complete evidence is provided about the truth of the conclusion made. Here, the agent uses specific and accurate premises that lead to a specific conclusion. An example of this logic can be seen in an expert system designed to suggest medicines to the patient. The agent gives the complete proof about the medicines suggested by it, like the particular medicines are suggested to a person because the person has so and so symptoms.

2) Inductive logic

In Inductive logic, the reasoning is done through a ‘bottom-up’ approach. What this means is that the agent here takes specific information and then generalizes it for the sake of complete understanding. An example of this can be seen in the natural language processing by an agent in which it sums up the words according to their category, i.e. verb, noun article, etc., and then infers the meaning of that sentence.

Propositional logic

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

Example:

1. a) It is Sunday.
2. b) The Sun rises from West (False proposition)
3. c) $3+3=7$ (False proposition)
4. d) 5 is a prime number.

Following are some basic facts about propositional logic:

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- A proposition formula which has both true and false values is called **Contingency**
- Statements which are questions, commands, or opinions are not propositions such as "**Where is Rohini**", "**How are you**", "**What is your name**", are not propositions.

Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

- a. **Atomic Propositions**
- b. **Compound propositions**

- **Atomic Proposition:** Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example:

1. a) $2+2$ is 4, it is an atomic proposition as it is a **true** fact.
2. b) "The Sun is cold" is also a proposition as it is a **false** fact.
 - **Compound proposition:** Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Example:

1. a) "It is raining today, and street is wet."
2. b) "Ankit is a doctor, and his clinic is in Mumbai."

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic. Example:
 - a. **All the girls are intelligent.**
 - b. **Some apples are sweet.**

Propositional logic has limited expressive power.

In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

First-Order Logic:

In the topic of Propositional logic, we have seen that how to represent statements using propositional logic. But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

- **"Some humans are intelligent", or**
- **"Sachin likes cricket."**

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

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

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

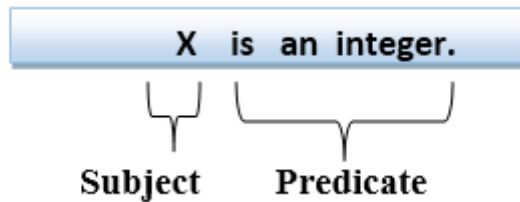
Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....

Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf,
Connectives	$\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$
Equality	$=$
Quantifier	\forall, \exists

First-order logic statements can be divided into two parts:

- **Subject:** Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
 - a. **Universal Quantifier, (for all, everyone, everything)**
 - b. **Existential quantifier, (for some, at least one).**

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol \forall , which resembles an inverted A.

If x is a variable, then $\forall x$ is read as:

- **For all x**
- **For each x**
- **For every x.**

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator \exists , which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

If x is a variable, then existential quantifier will be $\exists x$ or $\exists(x)$. And it will be read as:

- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Properties of Quantifiers:

- In universal quantifier, $\forall x\forall y$ is similar to $\forall y\forall x$.
- In Existential quantifier, $\exists x\exists y$ is similar to $\exists y\exists x$.
- $\exists x\forall y$ is not similar to $\forall y\exists x$.

Unification:

- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\Psi_1\sigma = \Psi_2\sigma$, then it can be expressed as **UNIFY(Ψ_1, Ψ_2)**.
- **Example: Find the MGU for Unify{King(x), King(John)}**

Let $\Psi_1 = \text{King}(x)$, $\Psi_2 = \text{King}(\text{John})$,

Substitution $\theta = \{\text{John}/x\}$ is a unifier for these atoms and applying this substitution, and both expressions will be identical.

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.
- It returns fail if the expressions do not match with each other.
- The substitution variables are called Most General Unifier or MGU.

E.g. Let's say there are two different expressions, **P(x, y), and P(a, f(z))**.

In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.

$P(x, y)$ (i)
 $P(a, f(z))$ (ii)

- Substitute x with a, and y with f(z) in the first expression, and it will be represented as **a/x** and f(z)/y.
- With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: [**a/x, f(z)/y**].

Conditions for Unification:

Following are some basic conditions for unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.

Unification Algorithm:

Algorithm: Unify(Ψ_1, Ψ_2)

Step. 1: If Ψ_1 or Ψ_2 is a variable or constant, then:

- If Ψ_1 or Ψ_2 are identical, then return NIL.
- Else if Ψ_1 is a variable,
 - then if Ψ_1 occurs in Ψ_2 , then return FAILURE
 - Else return $\{(\Psi_2/\Psi_1)\}$.
- Else if Ψ_2 is a variable,
 - If Ψ_2 occurs in Ψ_1 then return FAILURE,
 - Else return $\{(\Psi_1/\Psi_2)\}$.
- Else return FAILURE.

Step.2: If the initial Predicate symbol in Ψ_1 and Ψ_2 are not same, then return FAILURE.

Step. 3: IF Ψ_1 and Ψ_2 have a different number of arguments, then return FAILURE.

Step. 4: Set Substitution set(SUBST) to NIL.

Step. 5: For $i=1$ to the number of elements in Ψ_1 .

- a) Call Unify function with the i th element of Ψ_1 and i th element of Ψ_2 , and put the result into S.
- b) If S = failure then returns Failure
- c) If S \neq NIL then do,
 - a. Apply S to the remainder of both L1 and L2.
 - b. SUBST= APPEND(S, SUBST).

Step.6: Return SUBST.

Implementation of the Algorithm

Step.1: Initialize the substitution set to be empty.

Step.2: Recursively unify atomic sentences:

- a. Check for Identical expression match.
- b. If one expression is a variable v_i , and the other is a term t_i which does not contain variable v_i , then:
 - a. Substitute t_i / v_i in the existing substitutions
 - b. Add t_i / v_i to the substitution setlist.
 - c. If both the expressions are functions, then function name must be similar, and the number of arguments must be the same in both the expression.

For each pair of the following atomic sentences find the most general unifier (If exist).

1. Find the MGU of $\{p(f(a), g(Y))$ and $p(X, X)\}$

Sol: $S_0 \Rightarrow$ Here, $\Psi_1 = p(f(a), g(Y))$, and $\Psi_2 = p(X, X)$

SUBST $\theta = \{f(a) / X\}$

$S_1 \Rightarrow \Psi_1 = p(f(a), g(Y))$, and $\Psi_2 = p(f(a), f(a))$

SUBST $\theta = \{f(a) / g(y)\}$, **Unification failed.**

Unification is not possible for these expressions.

Resolution

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965.

Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.

Clause: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.

Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

The resolution inference rule:

The resolution rule for first-order logic is simply a lifted version of the propositional rule. Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

$$\frac{l_1 \vee \dots \vee l_k \quad m_1 \vee \dots \vee m_n}{\text{SUBST}(\theta, l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)}$$

Where l_i and m_j are complementary literals.

This rule is also called the **binary resolution rule** because it only resolves exactly two literals.

Example:

We can resolve two clauses which are given below:

[Animal (g(x) \vee Loves (f(x), x)] and [\neg Loves(a, b) \vee \neg Kills(a, b)]

Where two complimentary literals are: **Loves (f(x), x) and \neg Loves (a, b)**

These literals can be unified with unifier $\theta = [a/f(x), \text{ and } b/x]$, and it will generate a resolvent clause:

[Animal (g(x) \vee \neg Kills(f(x), x)].

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

- a. **John likes all kind of food.**
 - b. **Apple and vegetable are food**
 - c. **Anything anyone eats and not killed is food.**
 - d. **Anil eats peanuts and still alive**
 - e. **Harry eats everything that Anil eats.**
- Prove by resolution that:**
- f. **John likes peanuts.**

Step-1: Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.

- a. $\forall x: \text{food}(x) \rightarrow \text{likes}(\text{John}, x)$
- b. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
- c. $\forall x \forall y: \text{eats}(x, y) \wedge \neg \text{killed}(x) \rightarrow \text{food}(y)$
- d. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$.
- e. $\forall x : \text{eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$
- f. $\forall x: \neg \text{killed}(x) \rightarrow \text{alive}(x)$ } **added predicates.**
- g. $\forall x: \text{alive}(x) \rightarrow \neg \text{killed}(x)$ }
- h. $\text{likes}(\text{John}, \text{Peanuts})$

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

- o **Eliminate all implication (\rightarrow) and rewrite**

- a. $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
- b. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
- c. $\forall x \forall y \neg [\text{eats}(x, y) \wedge \neg \text{killed}(x)] \vee \text{food}(y)$
- d. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
- e. $\forall x \neg \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
- f. $\forall x \neg [\neg \text{killed}(x)] \vee \text{alive}(x)$
- g. $\forall x \neg \text{alive}(x) \vee \neg \text{killed}(x)$
- h. $\text{likes}(\text{John}, \text{Peanuts})$.

Move negation (\neg) inwards and rewrite

- . $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
- a. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
- b. $\forall x \forall y \neg \text{eats}(x, y) \vee \text{killed}(x) \vee \text{food}(y)$

- c. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
- d. $\forall x \neg \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
- e. $\forall x \neg \text{killed}(x) \vee \text{alive}(x)$
- f. $\forall x \neg \text{alive}(x) \vee \neg \text{killed}(x)$
- g. $\text{likes}(\text{John}, \text{Peanuts})$.

Rename variables or standardize variables

- . $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
- a. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
- b. $\forall y \forall z \neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$
- c. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
- d. $\forall w \neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$
- e. $\forall g \neg \text{killed}(g) \vee \text{alive}(g)$
- f. $\forall k \neg \text{alive}(k) \vee \neg \text{killed}(k)$
- g. $\text{likes}(\text{John}, \text{Peanuts})$.

Eliminate existential instantiation quantifier by elimination.

In this step, we will eliminate existential quantifier \exists , and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

Drop Universal quantifiers.

In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

- . $\neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
- a. $\text{food}(\text{Apple})$
- b. $\text{food}(\text{vegetables})$
- c. $\neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$
- d. $\text{eats}(\text{Anil}, \text{Peanuts})$
- e. $\text{alive}(\text{Anil})$
- f. $\neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$
- g. $\text{killed}(g) \vee \text{alive}(g)$
- h. $\neg \text{alive}(k) \vee \neg \text{killed}(k)$

i. likes(John, Peanuts).

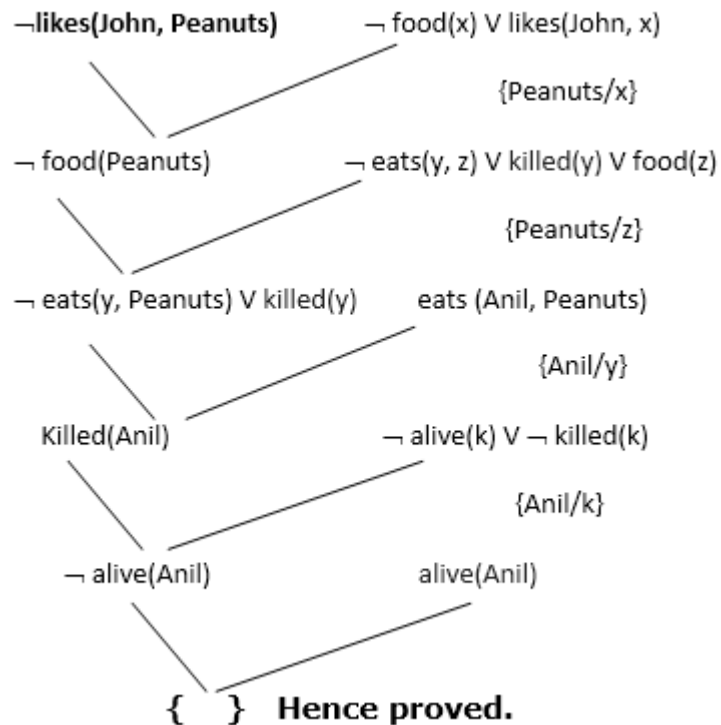
- o **Distribute conjunction \wedge over disjunction \neg .**
This step will not make any change in this problem.

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as \neg likes(John, Peanuts)

Step-4: Draw Resolution graph:

Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

D.N.R.COLLEGE(AUTONOMOUS):BHIMAVARAM
M.C.A DEPARTMENT



ARTIFICIAL INTELLIGENCE

Presented by

L.SOWJANYA

UNIT III

Acting Under Uncertainty, Basic Probability Notation, The Axioms of Probability, Inference Using Full Joint Distribution, Independence, Bayes Rule and Its Use, Other Approaches To Uncertain Reasoning: Dempster Shafer Theory, Fuzzy Sets and Fuzzy Logic
Combining Beliefs Under Uncertainty, The Basis of Utility Theory, Utility Functions, Multi Attribute Utility Functions, Decision Theoretic Expert Systems

UNIT-3

Reasoning Under Uncertainty: Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write $A \rightarrow B$, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

All of the time, agents are forced to make decisions based on incomplete information. Even when an agent senses the world to find out more information, it rarely finds out the exact state of the world. A robot does not know exactly where an object is. A doctor does not know exactly what is wrong with a patient. A teacher does not know exactly what a student understands. When intelligent agents must make decisions, they have to use whatever information they have. This chapter considers reasoning under uncertainty: determining what is true in the world based on observations of the world. basis for acting under uncertainty, where the agent must make decisions about what action to take even though it cannot precisely predict the outcomes of its actions. This chapter starts with probability, shows how to represent the world by making appropriate independence assumptions, and shows how to reason with such representations.

Causes of uncertainty:

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

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

Probabilistic reasoning:

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.

In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Axioms of Probability Theory

Probability Theory provides us with the formal mechanisms and rules for manipulating propositions represented probabilistically. The following are the three axioms of probability theory:

- $0 \leq P(A=a) \leq 1$ for all a in sample space of A

- $P(\text{True})=1, P(\text{False})=0$
- $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

From these axioms we can show the following properties also hold:

- $P(\sim A) = 1 - P(A)$
- $P(A) = P(A \wedge B) + P(A \wedge \sim B)$
- $\text{Sum}\{P(A=a)\} = 1$, where the sum is over all possible values a in the sample space of A

Joint Probability Distribution

Given an application domain in which we have determined a sufficient set of random variables to encode all of the relevant information about that domain, we can completely specify all of the possible probabilistic information by constructing the **full joint probability distribution**, $P(V_1=v_1, V_2=v_2, \dots, V_n=v_n)$, which assigns probabilities to all possible combinations of values to all random variables.

For example, consider a domain described by three Boolean random variables, Bird, Flier, and Young. Then we can enumerate a table showing all possible interpretations and associated probabilities:

Bird Flier Young Probability

T	T	T	0.0
T	T	F	0.2
T	F	T	0.04
T	F	F	0.01
F	T	T	0.01
F	T	F	0.01
F	F	T	0.23
F	F	F	0.5

Notice that there are 8 rows in the above table representing the fact that there are 2^3 ways to assign values to the three Boolean variables. More generally, with n Boolean variables the table will be of size 2^n . And if n variables each had k possible values, then the table would be size k^n .

Also notice that the sum of the probabilities in the right column must equal 1 since we know that the set of all possible values for each variable are known. This means that for n Boolean random variables, the table has 2^n-1 values that must be determined to completely fill in the table.

If all of the probabilities are known for a full joint probability distribution table, then we can compute *any* probabilistic statement about the domain. For example, using the table above, we can compute

- $P(\text{Bird}=T) = P(B) = 0.0 + 0.2 + 0.04 + 0.01 = 0.25$
- $P(\text{Bird}=T, \text{Flier}=F) = P(B, \sim F) = P(B, \sim F, Y) + P(B, \sim F, \sim Y) = 0.04 + 0.01 = 0.05$

Conditional probability:

Conditional probability is a probability of occurring an event when another event has already happened.

Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

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

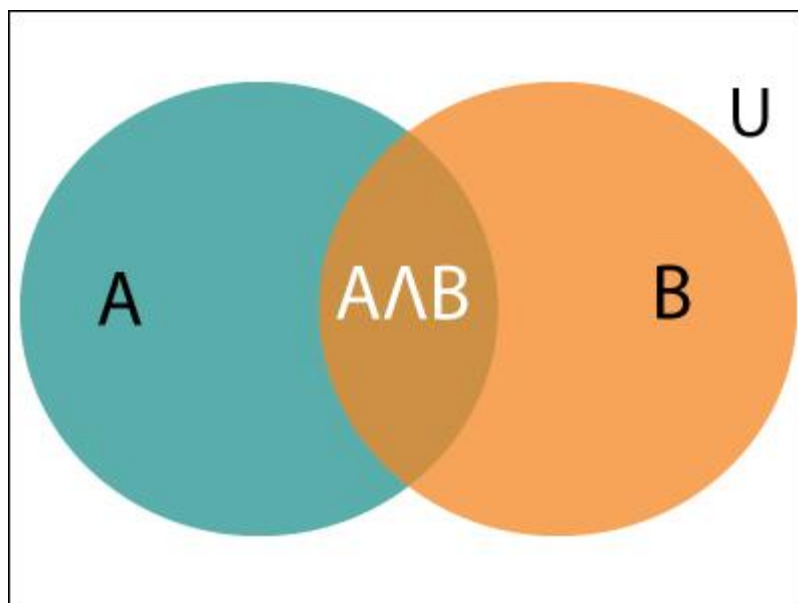
Where $P(A \cap B)$ = Joint probability of a and B

$P(B)$ = Marginal probability of B.

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

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

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



Example:

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

Solution:

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.

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

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

Bayes' theorem:

Bayes' theorem is also known as **Bayes' rule**, **Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.

In probability theory, it relates the conditional probability and marginal probabilities of two random events.

Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

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

Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

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

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

As from product rule we can write:

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

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

1. $P(A \cap B) = P(B|A) P(A)$

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

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)} \quad \dots(a)$$

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

It shows the simple relationship between joint and conditional probabilities. Here,

$P(A|B)$ is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

$P(B|A)$ is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.

$P(A)$ is called the **prior probability**, probability of hypothesis before considering the evidence

$P(B)$ is called **marginal probability**, pure probability of an evidence.

In the equation (a), in general, we can write $P(B) = \sum_{i=1}^k P(A_i) * P(B|A_i)$, hence the Bayes' rule can be written as:

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

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

Applying Bayes' rule:

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

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

Example-1:

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

Given Data:

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

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

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

$$P(a|b) = 0.8$$

$$P(b) = 1/30000$$

$P(a) = .02$

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8 + \left(\frac{1}{30000}\right)}{0.02} = 0.0013333333.$$

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

Application of Bayes' theorem in Artificial intelligence:

Following are some applications of Bayes' theorem:

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

Dempster Shafer Theory is given by Arthur P. Dempster in 1967 and his student Glenn Shafer in 1976. This theory is being released because of following reason:-

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

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

The **uncertainty in this model** is given by:-

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

For eg:-

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

To solve these there are the **following possibilities**:

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

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

Using the above example we can say :

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

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

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

For eg:-

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

{o, {a}, {b}, {c}, {a, b}, {b, c}, {a, c}, {a, b, c}} = 2³ elements.

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

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

Lets say $K = \{a, b, c\}$

$$\text{Bel}(K) = m(a) + m(b) + m(c) + m(a, b) + m(a, c) + m(b, c) + m(a, b, c)$$

Plausibility in K: It is the sum of masses of set that intersects with K. $1 - \text{Bel}(K)$
i.e; $\text{Pl}(K) = m(a) + m(b) + m(c) + m(a, b) + m(b, c) + m(a, c) + m(a, b, c)$

Characteristics of Dempster Shafer Theory:

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

Advantages:

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

Disadvantages:

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

Fuzzy Logic (FL) is a method of reasoning that resembles human reasoning. The approach of FL imitates the way of decision making in humans that involves all intermediate possibilities between digital values YES and NO.

The conventional logic block that a computer can understand takes precise input and produces a definite output as TRUE or FALSE, which is equivalent to human's YES or NO.

The inventor of fuzzy logic, Lotfi Zadeh, observed that unlike computers, the human decision making includes a range of possibilities between YES and NO, such as –

CERTAINLY YES
POSSIBLY YES
CANNOT SAY
POSSIBLY NO
CERTAINLY NO

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

Implementation

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

Why Fuzzy Logic?

Fuzzy logic is useful for commercial and practical purposes.

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

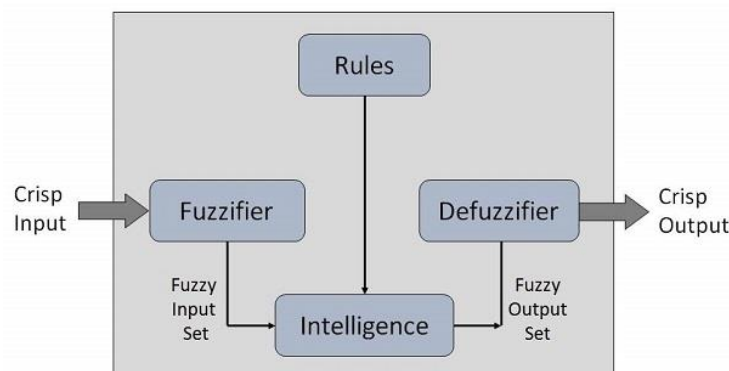
Fuzzy Logic Systems Architecture

It has four main parts as shown –

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

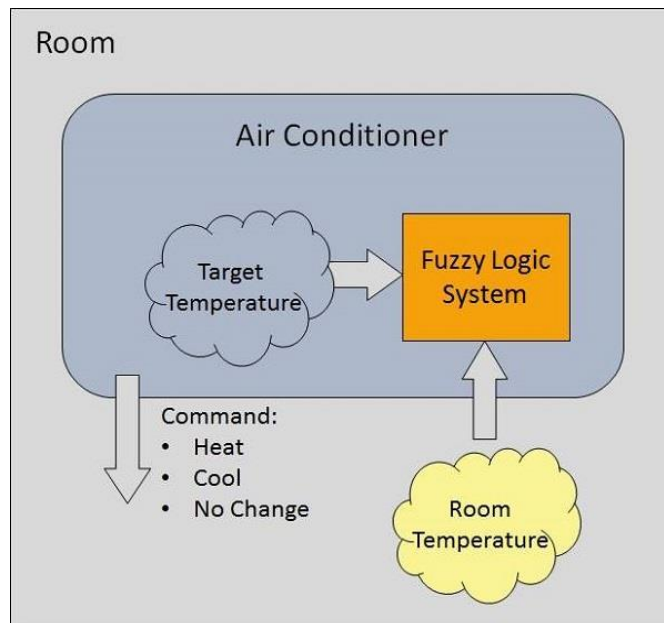
LP	x is Large Positive
MP	x is Medium Positive
S	x is Small
MN	x is Medium Negative
LN	x is Large Negative

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



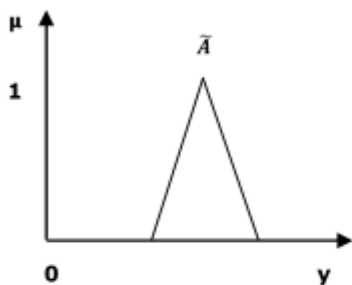
Example of a Fuzzy Logic System

Let us consider an air conditioning system with 5-level fuzzy logic system. This system adjusts the temperature of air conditioner by comparing the room temperature and the target temperature value.

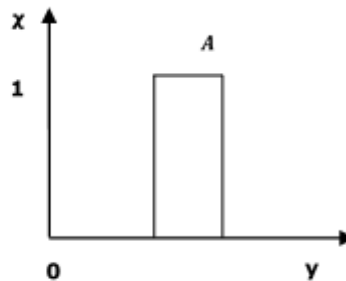


Fuzzy set:

Fuzzy sets can be considered as an extension and gross oversimplification of classical sets. It can be best understood in the context of set membership. Basically it allows partial membership which means that it contain elements that have varying degrees of membership in the set. From this, we can understand the difference between classical set and fuzzy set. Classical set contains elements that satisfy precise properties of membership while fuzzy set contains elements that satisfy imprecise properties of membership.



Membership Function of Fuzzy set \tilde{A}



Membership Function of classical set A

Utility Theory

Utility Theory is the discipline that lays out the foundation to create and evaluate Utility Functions. Typically, Utility Theory uses the notion of Expected Utility (EU) as a value that represents the average utility of all possible outcomes of a state, weighted by the probability that the outcome occurs.

The other key concept of Utility Theory is known as the Principle of Maximum Utility (MEU) which states that any rational agent should choose to maximize the agent's EU. The principle of MEU seems like an obvious way to make decisions until you start digging into it and run into all sorts of interesting questions. Why using the average utility after all? Why not to try to minimize the loss instead of maximizing utility? There are dozens of similar questions that challenge the principle of MEU. However, in order to validate the principle of MEU, we should go back to the laws of Utility Theory.

Utility Theory Axioms

There are six fundamental axioms that setup the foundation of Utility Theory. In order to explain those, let's pick a scenario in which you are having dinner at a restaurant and you are trying to decide between a salmon or a chicken dish. There are many factors that go into that simple decision. Which dish goes better with this gorgeous Montrachet we just ordered? How about the dessert? How would I feel if the chicken is overcooked? Is the salmon's portion too small?...Hopefully you got the point.

Utility theory assigns a probability to each one of those possible states that try to orchestrate decisions based on that. However, those decisions are governed by a group of six fundamental axioms: Orderability, Transitivity, Continuity, Substitutability, Monotonicity and Decomposability. Together these six axioms help to enforce the principle of MEU.

Utility function

Utility functions are a product of Utility Theory which is one of the disciplines that helps to address the challenges of building knowledge under uncertainty. Utility theory is often combined with probabilistic theory to create what we know as decision-theoretic agents. Conceptually, a decision-theoretic agent is an AI program that can make rational decisions based on what it believes and what it wants. Sounds rational, right? :)

Ok, let's get a bit more practical. In many AI scenarios, agents don't have the luxury of operating in an environment in which they know the final outcome of every possible state. Those agents operate under certain degree of uncertainty and need to rely on probabilities to quantify the outcome of possible states. That probabilistic function is what we call Utility Functions.

Multi-attribute utility function

In [decision theory](#), a **multi-attribute utility** function is used to represent the preferences of an agent over bundles of goods either under conditions of certainty about the results of any potential choice, or under conditions of uncertainty.

A person has to decide between two or more options. The decision is based on the *attributes* of the options.

The simplest case is when there is only one attribute, e.g.: money. It is usually assumed that all people prefer more money to less money; hence, the problem in this case is trivial: select the option that gives you more money.

In reality, there are two or more attributes. For example, a person has to select between two employment options: option A gives him \$12K per month and 20 days of vacation, while option B gives him \$15K per month and only 10 days of vacation. The person has to decide between (12K,20) and (15K,10). Different people may have different preferences. Under certain conditions, a person's preferences can be represented by a numeric function. The article [ordinal utility](#) describes some properties of such functions and some ways by which they can be calculated.

Another consideration that might complicate the decision problem is uncertainty. This complication exists even when there is a single attribute, e.g.: money. For example, option A might be a lottery with 50% chance to win \$2, while option B is to win \$1 for sure. The person has to decide between the lottery $\langle 2:0.5 \rangle$ and the lottery $\langle 1:1 \rangle$. Again, different people may have different preferences. Again, under certain conditions the preferences can be represented by a numeric function. Such functions are called [cardinal utility](#) functions. The article [Von Neumann–Morgenstern utility theorem](#) describes some ways by which they can be calculated.

The most general situation is that there are *both* multiple attributes *and* uncertainty. For example, option A may be a lottery with a 50% chance to win two apples and two bananas, while option B is to win two

bananas for sure. The decision is between $\langle(2,2):(0.5,0.5)\rangle$ and $\langle(2,0):(1,0)\rangle$. The preferences here can be represented by [cardinal utility](#) functions which take several variables (the attributes). Such functions are the focus of the current article.

The goal is to calculate a utility function which represents the person's preferences on lotteries of bundles. I.e, lottery A is preferred over lottery B if and only if the expectation of the function is higher under A than under B:

Assessing a multi-attribute cardinal utility function

If the number of possible bundles is finite, u can be constructed directly as explained by [von Neumann and Morgenstern](#) (VNM): order the bundles from least preferred to most preferred, assign utility 0 to the former and utility 1 to the latter, and assign to each bundle in between a utility equal to the probability of an equivalent lottery.

If the number of bundles is infinite, one option is to start by ignoring the randomness, and assess an [ordinal utility](#) function which represents the person's utility on *sure* bundles. I.e, a bundle x is preferred over a bundle y if and only if the function is higher for x than for y :

This function, in effect, converts the multi-attribute problem to a single-attribute problem: the attribute is v . Then, VNM can be used to construct the function

Note that u must be a positive monotone transformation of v . This means that there is a monotonically increasing function r , such that:

The problem with this approach is that it is not easy to assess the function r . When assessing a single-attribute cardinal utility function using VNM, we ask questions such as: "What probability to win \$2 is equivalent to \$1?". So to assess the function r , we have to ask a question such as: "What probability to win 2 units of value is equivalent to 1 value?". The latter question is much harder to answer than the former, since it involves "value", which is an abstract quantity.

A possible solution is to calculate n one-dimensional cardinal utility functions - one for each attribute. For

example, suppose there are two attributes: apples () and bananas (), both range between 0 and 99. Using VNM, we can calculate the following 1-dimensional utility functions:

- - a cardinal utility on apples when there are no bananas (the southern boundary of the domain);
- - a cardinal utility on bananas when apples are at their maximum (the eastern boundary of the domain).

Using linear transformations, scale the functions such that they have the same value on (99,0).

Then, for every bundle (a,b) , find an equivalent bundle (a bundle with the same v) which is either of the form $(a,0)$ or of the form $(0,b)$, and set its utility to the same number.

Often, certain independence properties between attributes can be used to make the construction of a utility function easier.

Agents and decision theory

Agents need to make decisions in situations of uncertainty and conflicting goals v Basic principle of decision theory: Maximization of expected utility v Decision-theoretic agents are based decision theory, and need knowledge of probability and utility v Here, we are concerned with "simple" (one shot) decisions, can be extended to sequential decisions.

Principle of Maximum Expected Utility (MEU)

Let $U(s)$ - Utility of state s , $RESULT(a)$ – Random variable whose values are possible outcome states of action a in current state, $P(RESULT(a) = s' | a, e)$ - Probability of outcome s' , as a result of doing action a in current state, and given agent's available evidence e of the world. Then the expected utility EU of a , given e is $EU_a = \sum_s P(RESULT(a) = s) U(s)$

Agent should select a that maximizes EU

Problems with applying MEU

Often difficult to formulate problem completely, and required computation can be prohibitive v Knowing state of the world requires perception, learning, representation and inference

Computing $P(RESULT(a) | a, e)$ requires complete causal model and NP-complete belief net updating

Computing utility $U(s')$ may require search or planning since agent needs to know how to get to a state before its utility can be assessed.

Decision networks

Decision networks (also called influence diagrams) are a general mechanism for making rational decisions v Decision networks combine belief networks with nodes for actions and utilities, and can represent

- Information about agent's current state

- Agent's possible actions

- States that will follow from actions

- Utilities of these states

Therefore, decision networks provide a substrate for implementing rational, utility-based agents

Node types in decision networks

Chance nodes (ovals) Represent random variables (as in belief networks), with associated conditional probability table (CPT) indexed by states of parent nodes (decisions or other chance nodes)

Decision nodes (rectangles) Represent points where the decision maker has choice of actions to make

Utility nodes (diamonds) Represent the agent's utility function, with parents all nodes that directly influence utility.

Evaluating decision networks

Set the evidence variables (chance nodes with known values) for the current state v

For each possible value of the decision node

Set decision node to that value (from now on, it behaves like a chance node that has been set as an evidence variable)

Calculate posterior probabilities for parent nodes of the utility node, using standard probabilistic inference methods

Calculate resulting utility for the action

Return the action with the highest utility .

Decision analysis vs. expert systems

Decision analysis (application of decision theory)

Focus on making decisions

Defines possible actions and outcomes with preferences

Roles • Decision maker states preferences • Decision analyst specifies problem

Expert systems (“classical” rule-based systems)

Focus on answering questions

Defines heuristic associations between evidence & answers

Roles • Domain expert provides heuristic knowledge • Knowledge engineer elicits & encodes knowledge in rules.

Decision-theoretic expert systems

Decision-theoretic expert systems

Inclusion of decision networks in expert system frameworks

Advantages

Make expert preferences explicit

Automate action selection in addition to inference

Avoid confusing likelihood with importance.

• Common pitfall in expert systems: Conclusions are ranked in terms of likelihood, disregarding rare, but dangerous conclusion

Availability of utility information helps in knowledge engineering process.

Procedural versus Declarative Knowledge

We have discussed various search techniques in previous units. Now we would consider a set of rules that represent,

1. Knowledge about relationships in the world and
2. Knowledge about how to solve the problem using the content of the rules.

Procedural vs Declarative Knowledge

Procedural Knowledge

• A representation in which the control information that is necessary to use the knowledge is embedded in the knowledge itself for e.g. computer programs, directions, and recipes; these indicate specific use or implementation;

• The real difference between declarative and procedural views of knowledge lies in where control information reside.

For example, consider the following

Man (Marcus)

Man (Caesar)

Person (Cleopatra)

$\forall x: \text{Man}(x) \rightarrow \text{Person}(x)$

Now, try to answer the question. *?Person(y)*

The knowledge base justifies any of the following answers.

Y=Marcus

Y=Caesar

$Y = \text{Cleopatra}$

- We get more than one value that satisfies the predicate.
- If only one value needed, then the answer to the question will depend on the order in which the assertions examined during the search for a response.
- If the assertions declarative then they do not themselves say anything about how they will be examined. In case of procedural representation, they say how they will examine.

Declarative Knowledge

- A statement in which knowledge specified, but the use to which that knowledge is to be put is not given.
- For example, laws, people's name; these are the facts which can stand alone, not dependent on other knowledge;
- So to use declarative representation, we must have a program that explains what is to do with the knowledge and how.
- For example, a set of logical assertions can combine with a resolution theorem prove to give a complete program for solving problems but in some cases, the logical assertions can view as a program rather than data to a program.
- Hence the implication statements define the legitimate reasoning paths and automatic assertions provide the starting points of those paths.
- These paths define the execution paths which is similar to the 'if then else' in traditional programming.
- So logical assertions can view as a procedural representation of knowledge.

Logic Programming – Representing Knowledge Using Rules

- Logic programming is a programming paradigm in which logical assertions viewed as programs.
- These are several logic programming systems, PROLOG is one of them.
- ***A PROLOG program consists of several logical assertions where each is a horn clause i.e. a clause with at most one positive literal.***
- Ex : $P, P \vee Q, P \rightarrow Q$
- The facts are represented on Horn Clause for two reasons.
 1. Because of a uniform representation, a simple and efficient interpreter can write.
 2. The logic of Horn Clause decidable.
- Also, The first two differences are the fact that PROLOG programs are actually sets of Horn clause that have been transformed as follows:-
 1. If the Horn Clause contains no negative literal then leave it as it is.
 2. Also, Otherwise rewrite the Horn clauses as an implication, combining all of the negative literals into the antecedent of the implications and the single positive literal into the consequent.
- Moreover, this procedure causes a clause which originally consisted of a disjunction of literals (one of them was positive) to be transformed into a single implication whose antecedent is a conjunction universally quantified.
- But when we apply this transformation, any variables that occurred in negative literals and so now occur in the antecedent become existentially quantified, while the variables in the consequent are still universally quantified.

For example the PROLOG clause $P(x) :- Q(x, y)$ is equal to logical expression $\forall x: \exists y: Q(x, y) \rightarrow P(x)$.

- The difference between the logic and PROLOG representation is that the PROLOG interpretation has a fixed control strategy. And so, the assertions in the PROLOG program define a particular search path to answer any question.
- But, the logical assertions define only the set of answers but not about how to choose among those answers if there is more than one.

Consider the following example:

1. Logical representation

$\forall x : \text{pet}(x) \wedge \text{small}(x) \rightarrow \text{apartmentpet}(x)$

$\forall x : \text{cat}(x) \wedge \text{dog}(x) \rightarrow \text{pet}(x)$

$\forall x : poodle(x) \rightarrow dog(x) \wedge small(x)$
poodle (fluffy)

2. Prolog representation

apartmentpet(x) : pet(x), small(x)

pet(x) : cat(x)

pet(x) : dog(x)

dog(x) : poodle(x)

small(x) : poodle(x)

poodle (fluffy)

Forward versus Backward Reasoning

Forward versus Backward Reasoning

A search procedure must find a path between initial and goal states.

There are two directions in which a search process could proceed.

The two types of search are:

1. Forward search which starts from the start state
2. Backward search that starts from the goal state

The production system views the forward and backward as symmetric processes.

Consider a game of playing 8 puzzles. The rules defined are

Square 1 empty and square 2 contains tile n. \rightarrow

Also, Square 2 empty and square 1 contains the tile n.

Square 1 empty Square 4 contains tile n. $\rightarrow \square \square$

Also, Square 4 empty and Square 1 contains tile n.

We can solve the problem in 2 ways:

1. Reason forward from the initial state

- Step 1. Begin building a tree of move sequences by starting with the initial configuration at the root of the tree.
- Step 2. Generate the next level of the tree by finding all rules ***whose left-hand side matches*** against the root node. The right-hand side is used to create new configurations.
- Step 3. Generate the next level by considering the nodes in the previous level and applying it to all rules whose left-hand side match.

2. Reasoning backward from the goal states:

- Step 1. Begin building a tree of move sequences by starting with the goal node configuration at the root of the tree.
- Step 2. Generate the next level of the tree by finding all rules ***whose right-hand side matches*** against the root node. The left-hand side used to create new configurations.
- Step 3. Generate the next level by considering the nodes in the previous level and applying it to all rules whose right-hand side match.
- So, The same rules can use in both cases.
- Also, In forwarding reasoning, the left-hand sides of the rules matched against the current state and right sides used to generate the new state.
- Moreover, In backward reasoning, the right-hand sides of the rules matched against the current state and left sides are used to generate the new state.

There are four factors influencing the type of reasoning. They are,

1. Are there more possible start or goal state? We move from smaller set of sets to the length.

2. In what direction is the branching factor greater? We proceed in the direction with the lower branching factor.
3. Will the program be asked to justify its reasoning process to a user? If, so then it is selected since it is very close to the way in which the user thinks.
4. What kind of event is going to trigger a problem-solving episode? If it is the arrival of a new factor, the forward reasoning makes sense. If it is a query to which a response is desired, backward reasoning is more natural.

Example 1 of Forward versus Backward Reasoning

- It is easier to drive from an unfamiliar place from home, rather than from home to an unfamiliar place. Also, If you consider a home as starting place an unfamiliar place as a goal then we have to backtrack from unfamiliar place to home.

Example 2 of Forward versus Backward Reasoning

- Consider a problem of symbolic integration. Moreover, The problem space is a set of formulas, which contains integral expressions. Here START is equal to the given formula with some integrals. GOAL is equivalent to the expression of the formula without any integral. Here we start from the formula with some integrals and proceed to an integral free expression rather than starting from an integral free expression.

Example 3 of Forward versus Backward Reasoning

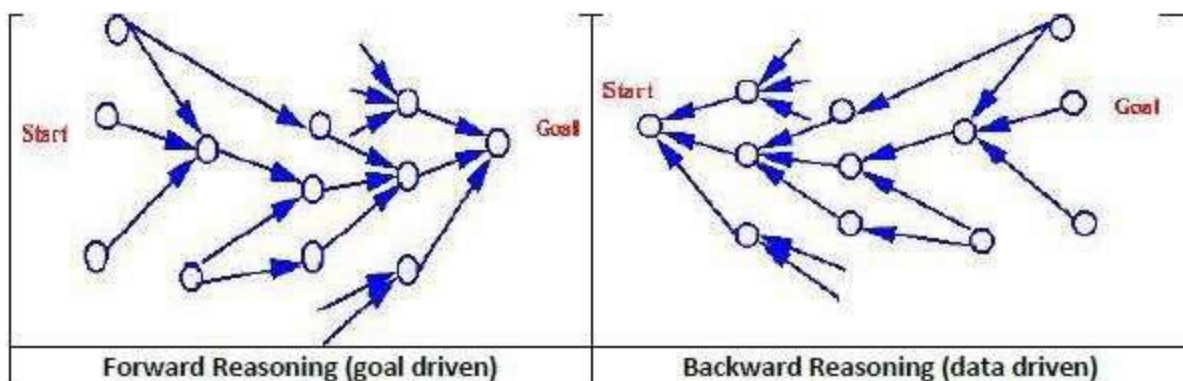
- The third factor is nothing but deciding whether the reasoning process can justify its reasoning. If it justifies then it can apply. For example, doctors are usually unwilling to accept any advice from diagnostics process because it cannot explain its reasoning.

Example 4 of Forward versus Backward Reasoning

- Prolog is an example of backward chaining rule system. In Prolog rules restricted to Horn clauses. This allows for rapid indexing because all the rules for deducing a given fact share the same rule head. Rules matched with unification procedure. Unification tries to find a set of bindings for variables to equate a sub-goal with the head of some rule. Rules in the Prolog program matched in the order in which they appear.

Combining Forward and Backward Reasoning

- Instead of searching either forward or backward, you can search both simultaneously.
- Also, That is, start forward from a starting state and backward from a goal state simultaneously until the paths meet.
- This strategy called Bi-directional search. The following figure shows the reason for a Bidirectional search to be ineffective.



Forward versus Backward Reasoning

- Also, The two searches may pass each other resulting in more work.
- Based on the form of the rules one can decide whether the same rules can apply to both forward and backward reasoning.
- Moreover, If left-hand side and right of the rule contain pure assertions then the rule can reverse.
- And so the same rule can apply to both types of reasoning.
- If the right side of the rule contains an arbitrary procedure then the rule cannot reverse.
- So, In this case, while writing the rule the commitment to a direction of reasoning must make.

Symbolic Reasoning under Uncertainty

Symbolic Reasoning

- The reasoning is the act of deriving a conclusion from certain properties using a given methodology.
- The reasoning is a process of thinking; reasoning is logically arguing; reasoning is drawing the inference.
- □ *When a system is required to do something, that it has not been explicitly told how to do, it must reason. It must figure out what it needs to know from what it already knows.*
- Many types of Reasoning have been identified and recognized, but many questions regarding their logical and computational properties still remain controversial.
- The popular methods of Reasoning include abduction, induction, model-based, explanation and confirmation. All of them are intimately related to problems of belief revision and theory development, knowledge absorption, discovery, and learning.

Logical Reasoning

- Logic is a language for reasoning. It is a collection of rules called Logic arguments, we use when doing logical reasoning.
- The logic reasoning is the process of drawing conclusions from premises using rules of inference.
- The study of logic divided into formal and informal logic. The formal logic is sometimes called symbolic logic.
- Symbolic logic is the study of symbolic abstractions (construct) that capture the formal features of logical inference by a formal system.
- The formal system consists of two components, a formal language plus a set of inference rules.
- The formal system has axioms. Axiom is a sentence that is always true within the system.
- Sentences derived using the system's axioms and rules of derivation called theorems.
- The Logical Reasoning is of our concern in AI.

Approaches to Reasoning

- There are three different approaches to reasoning under uncertainties.
 1. Symbolic reasoning
 2. Statistical reasoning
 3. Fuzzy logic reasoning

Symbolic Reasoning

- The basis for intelligent mathematical software is the integration of the “power of symbolic mathematical tools” with the suitable “proof technology”.
- Mathematical reasoning enjoys a property called monotonicity, that says, “If a conclusion follows from given premises A, B, C... then it also follows from any larger set of premises, as long as the original premises A, B, C.. Included.”
- Moreover, Human reasoning is not monotonic.

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

Formal Logic

Moreover, The Formal logic is the study of inference with purely formal content, i.e. where content made explicit.

Examples – Propositional logic and Predicate logic.

- Here the logical arguments are a set of rules for manipulating symbols. The rules are of two types,
 1. Syntax rules: say how to build meaningful expressions.
 2. Inference rules: say how to obtain true formulas from other true formulas.
- Moreover, Logic also needs semantics, which says how to assign meaning to expressions.

Uncertainty in Reasoning

- The world is an uncertain place; often the Knowledge is imperfect which causes uncertainty.
- So, Therefore reasoning must be able to operate under uncertainty.
- Also, AI systems must have the ability to reason under conditions of uncertainty.

Monotonic Reasoning

- A reasoning process that moves in one direction only.
- Moreover, The number of facts in the knowledge base is always increasing.
- The conclusions derived are valid deductions and they remain so.

A monotonic logic cannot handle

1. Reasoning by default: because consequences may derive only because of lack of evidence to the contrary.
2. Abductive reasoning: because consequences only deduced as most likely explanations.
3. Belief revision: because new knowledge may contradict old beliefs.

Introduction to Nonmonotonic Reasoning

Non-monotonic Reasoning

The definite clause logic is **monotonic** in the sense that anything that could be concluded before a clause is added can still be concluded after it is added; adding knowledge does not reduce the set of propositions that can be derived.

A logic is **non-monotonic** if some conclusions can be invalidated by adding more knowledge. The logic of definite clauses with negation as failure is non-monotonic. Non-monotonic reasoning is useful for representing defaults. A **default** is a rule that can be used unless it overridden by an exception.

For example, to say that b is normally true if c is true, a knowledge base designer can write a rule of the form

$$b \leftarrow c \wedge \sim ab_a.$$

where ab_a is an atom that means abnormal with respect to some aspect a . Given c , the agent can infer b unless it is told ab_a . Adding ab_a to the knowledge base can prevent the conclusion of b .

Rules that imply ab_a can be used to prevent the default under the conditions of the body of the rule.

Example 5.27: Suppose the purchasing agent is investigating purchasing holidays. A resort may be adjacent to a beach or away from a beach. This is not symmetric; if the resort was adjacent to a beach, the knowledge provider would specify this. Thus, it is reasonable to have the clause *away_from_beach* $V \sim$ *on_beach*.

This clause enables an agent to infer that a resort is away from the beach if the agent is not told it is adjacent to a beach.

A **cooperative system** tries to not mislead. If we are told the resort is on the beach, we would expect that resort users would have access to the beach. If they have access to a beach, we would expect them to be able to swim at the beach. Thus, we would expect the following defaults:

$beach_access \sqsubset on_beach \wedge \sim ab_{beach_access}.$
 $swim_at_beach \sqsubset beach_access \wedge \sim ab_{swim_at_beach}.$

A cooperative system would tell us if a resort on the beach has no beach access or if there is no swimming. We could also specify that, if there is an enclosed bay and a big city, then there is no swimming, by default:

$ab_{swim_at_beach} \sqsubset enclosed_bay \wedge big_city \wedge \sim ab_{no_swimming_near_city}.$

We could say that British Columbia is abnormal with respect to swimming near cities:

$ab_{no_swimming_near_city} \leftarrow in_BC \wedge \sim ab_{BC_beaches}.$

Given only the preceding rules, an agent infers *away_from_beach*. If it is then told *on_beach*, it can no longer infer *away_from_beach*, but it can now infer *beach_access* and *swim_at_beach*. If it is also told *enclosed_bay* and *big_city*, it can no longer infer *swim_at_beach*. However, if it is then told *in_BC*, it can then infer *swim_at_beach*.

By having defaults of what is normal, a user can interact with the system by telling it what is abnormal, which allows for economy in communication. The user does not have to state the obvious.

One way to think about non-monotonic reasoning is in terms of **arguments**. The rules can be used as components of arguments, in which the negated abnormality gives a way to undermine arguments. Note that, in the language presented, only positive arguments exist that can be undermined. In more general theories, there can be positive and negative arguments that attack each other.

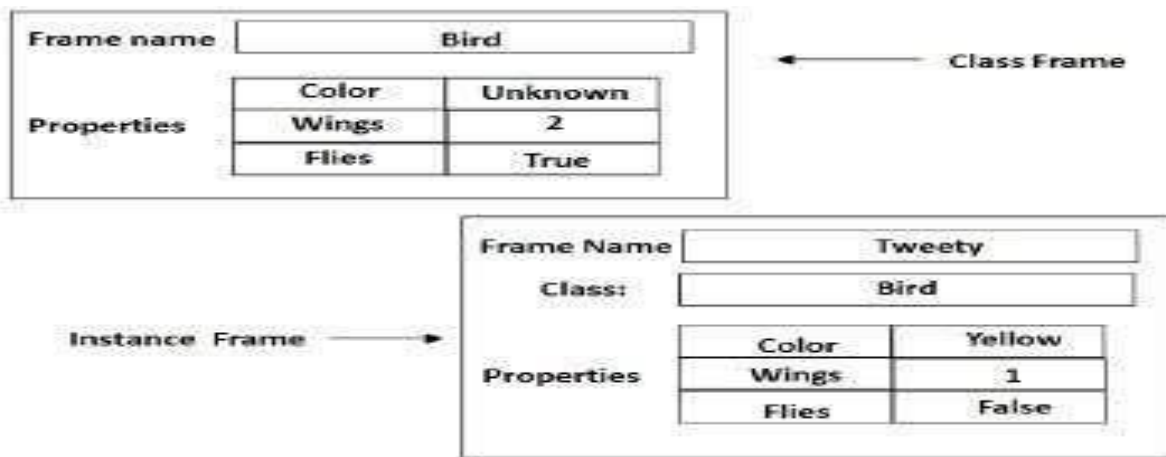
Weak Slot and Filler Structures

Evolution Frames

- As seen in the previous example, there are certain problems which are difficult to solve with Semantic Nets.
- Although there is no clear distinction between a semantic net and frame system, more structured the system is, more likely it is to be termed as a frame system.
- A frame is a collection of attributes (called slots) and associated values that describe some entities in the world. Sometimes a frame describes an entity in some absolute sense;
- Sometimes it represents the entity from a particular point of view only.
- A single frame taken alone is rarely useful; we build frame systems out of collections of frames that connected to each other by virtue of the fact that the value of an attribute of one frame may be another frame.

Frames as Sets and Instances

- The set theory is a good basis for understanding frame systems.
- Each frame represents either a class (a set) or an instance (an element of class)
- Both *isa* and *instance* relations have inverse attributes, which we call subclasses & all instances.
- As a class represents a set, there are 2 kinds of attributes that can be associated with it.
 1. Its own attributes &
 2. Attributes that are to be inherited by each element of the set.



Frames as Sets and Instances

- Sometimes, the difference between a set and an individual instance may not be clear.
- Example: Team India is an instance of the class of Cricket Teams and can also think of as the set of players.
- Now the problem is if we present Team India as a subclass of Cricket teams, then Indian players automatically become part of all the teams, which is not true.
- So, we can make Team India a subclass of class called Cricket Players.
- To do this we need to differentiate between regular classes and meta-classes.
- Regular Classes are those whose elements are individual entities whereas Meta-classes are those special classes whose elements are themselves, classes.
- The most basic meta-class is the class *CLASS*.
- It represents the set of all classes.
- All classes are instances of it, either directly or through one of its subclasses.
- The class *CLASS* introduces the attribute cardinality, which is to be inherited by all instances of *CLASS*. Cardinality stands for the number.

Other ways of Relating Classes to Each Other

- We have discussed that a class1 can be a subset of class2.
- If Class2 is a meta-class then Class1 can be an instance of Class2.
- Another way is the *mutually-disjoint-with* relationship, which relates a class to one or more other classes that guaranteed to have no elements in common with it.
- Another one is, *is-covered-by* which relates a class to a set of subclasses, the union of which is equal to it.
- If a class is-covered-by a set S of mutually disjoint classes, then S called a partition of the class.

Slots as Full-Fledged Objects (Frames)

Till now we have used attributes as slots, but now we will represent attributes explicitly and describe their properties.

Some of the properties we would like to be able to represent and use in reasoning include,

- The class to which the attribute can attach.
- Constraints on either the type or the value of the attribute.
- A default value for the attribute. Rules for inheriting values for the attribute.
- To be able to represent these attributes of attributes, we need to describe attributes (slots) as frames.
- These frames will organize into an *isa* hierarchy, just as any other frames, and that hierarchy can then used to support inheritance of values for attributes of slots.
- Now let us formalize what is a slot. A slot here is a relation.

- It maps from elements of its domain (the classes for which it makes sense) to elements of its range (its possible values).
- A relation is a set of ordered pairs.
- Thus it makes sense to say that relation R1 is a subset of another relation R2.
- In that case, R1 is a specialization of R2. Since a slot is a set, the set of all slots, which we will call SLOT, is a meta-class.
- Its instances are slots, which may have sub-slots.

Frame Example

In this example, the frames Person, Adult-Male, ML-Baseball-Player (corresponding to major league baseball players), Pitcher, and ML-Baseball-Team (for major league baseball team) are all classes.

Person	
isa :	Mammal
cardinality :	6,000,000,000
* handed :	Right
Adult-Male	
isa :	Person
cardinality :	2,000,000,000
* height :	5-10
ML-Baseball-Player	
isa :	Adult-Male
cardinality :	624
* height :	6-1
* bats :	equal to handed
* batting-average :	.252
* team :	
* uniform-color :	
Fielder	
isa :	ML-Baseball-Player
cardinality :	376
* batting-average :	.262
Pee-Wee-Reese	
instance :	Fielder
height :	5-10
bats :	Right
batting-average :	.309
team :	Brooklyn-Dodgers
uniform-color :	Blue
ML-Baseball-Team	
isa :	Team
cardinality :	26
* team-size :	24
* manager :	
Brooklyn-Dodgers	
instance :	ML-Baseball-Team
team-size :	24
manager :	Leo-Durocher
players :	{Pee-Wee-Reese,...}

- The frames Pee-Wee-Reese and Brooklyn-Dodgers are instances.
- The *isa* relation that we have been using without a precise definition is, in fact, the subset relation. The set of adult males is a subset of the set of people.
- The set of major league baseball players subset of the set of adult males, and so forth.
- Our instance relation corresponds to the relation element-of Pee Wee Reese is an element of the set of fielders.
- Thus he is also an element of all of the supersets of fielders, including major league baseball players and people. The transitivity of *is-a* follows directly from the transitivity of the subset relation.
- Both the *isa* and instance relations have inverse attributes, which we call subclasses and all instances.
- Because a class represents a set, there are two kinds of attributes that can associate with it.
- Some attributes are about the set itself, and some attributes are to inherited by each element of the set.
- We indicate the difference between these two by prefixing the latter with an asterisk (*).
- For example, consider the class ML-Baseball-Player, we have shown only two properties of it as a set: It a subset of the set of adult males. And it has cardinality 624.
- We have listed five properties that all major league baseball players have (height, bats, batting average, team, and uniform-color), and we have specified default values for the first three of them.
- By providing both kinds of slots, we allow both classes to define a set of objects and to describe a prototypical object of the set.
- Frames are useful for representing objects that are typical of stereotypical situations.
- The situation like the structure of complex physical objects, visual scenes, etc.
- A commonsense knowledge can represent using default values if no other value exists. Commonsense is generally used in the absence of specific knowledge.

Semantic Nets

- Inheritance property can represent using **isa** and **instance**
 - Monotonic Inheritance can perform substantially more efficiently with such structures than with pure logic, and non-monotonic inheritance is also easily supported.
 - The reason that makes Inheritance easy is that the knowledge in slot and filler systems is structured as a set of entities and their attributes.
- These structures turn out to be useful as,
- It indexes assertions by the entities they describe. As a result, retrieving the value for an attribute of an entity is fast.
 - Moreover, It makes easy to describe properties of relations. To do this in a purely logical system requires higher-order mechanisms.
 - It is a form of object-oriented programming and has the advantages that such systems normally include modularity and ease of viewing by people.
- Here we would describe two views of this kind of structure – Semantic Nets & Frames.

Semantic Nets

- There are different approaches to knowledge representation include semantic net, frames, and script.
- The semantic net describes both objects and events.
- In a semantic net, information represented as a set of nodes connected to each other by a set of labelled arcs, which represents relationships among the nodes.
- It is a directed graph consisting of vertices which represent concepts and edges which represent semantic relations between the concepts.
- It is also known as associative net due to the association of one node with other.
- The main idea is that the meaning of the concept comes from the ways in which it connected to other concepts.

- We can use inheritance to derive additional relations.

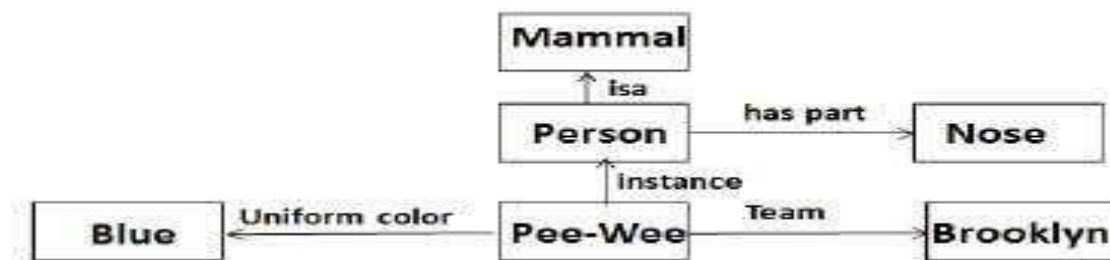


Figure: A Semantic Network

Intersection Search Semantic Nets

- We try to find relationships among objects by spreading activation out from each of two nodes. And seeing where the activation meets.
- Using this we can answer the questions like, what is the relation between India and Blue.
- It takes advantage of the entity-based organization of knowledge that slot and filler representation provides.

Representing Non-binary Predicates Semantic Nets

- Simple binary predicates like $isa(Person, Mammal)$ can represent easily by semantic nets but other non-binary predicates can also represent by using general-purpose predicates such as *isa* and *instance*.
- Three or even more place predicates can also convert to a binary form by creating one new object representing the entire predicate statement and then introducing binary predicates to describe a relationship to this new object.

Introduction to Strong Slot and Filler Structures

- The main problem with semantic networks and frames is that they lack formality; there is no specific guideline on how to use the representations.
- In frame when things change, we need to modify all frames that are relevant – this can be time-consuming.
- Strong slot and filler structures typically represent links between objects according to more rigid rules, specific notions of what types of object and relations between them are provided and represent knowledge about common situations.
- Moreover, We have types of strong slot and filler structures:
 1. Conceptual Dependency (CD)
 2. Scripts
 3. Cyc

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 the 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. Moreover, It has used by many programs that portend to understand English (MARGIE, SAM, PAM).

Conceptual Dependency (CD) provides:

- A structure into which nodes representing information can be placed.
- Also, A specific set of primitives.

- 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 represented.
- Moreover, The actions are built up from a set of primitive acts which can modify by tense.

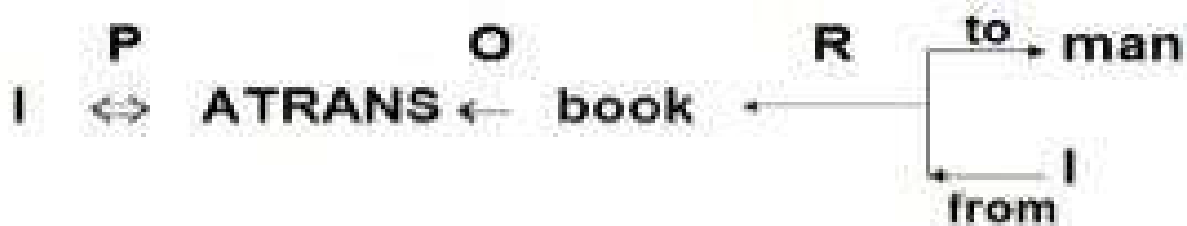
CD is based on events and actions. Every event (if applicable) has:

- an ACTOR or an ACTION performed by the Actor
- Also, an OBJECT that the action performs on
- A DIRECTION in which that action is oriented

These are represented as slots and fillers. In English sentences, many of these attributes left out.

A Simple Conceptual Dependency Representation

For the sentences, “I have a book to the man” CD representation is as follows:



Where the symbols have the following meaning.

- Arrows indicate directions of dependency.
- Moreover, the double arrow indicates the two-way link between actor and action.
- O — for the object case relation
- R – for the recipient case relation
- P – for past tense
- D – destination

Primitive Acts of Conceptual Dependency Theory

ATRANS

- Transfer of an abstract relationship (i.e. give)

PTRANS

- Transfer of the physical location of an object (e.g., go)

PROPEL

- Also, Application of physical force to an object (e.g. push)

MOVE

- Moreover, Movement of a body part by its owner (e.g. kick)

GRASP

- Grasping of an object by an action (e.g. throw)

INGEST

- Ingesting of an object by an animal (e.g. eat)

EXPEL

- Expulsion of something from the body of an animal (e.g. cry)

MTRANS

- Transfer of mental information (e.g. tell)

MBUILD

- Building new information out of old (e.g. decide)

SPEAK

- Producing of sounds (e.g. say)

ATTEND

- Focusing of a sense organ toward a stimulus (e.g. listen)

There are four conceptual categories. These are,

ACT

- Actions {one of the CD primitives}

PP

- Also, Objects {picture producers}

AA

- Modifiers of actions {action aiders}

PA

- Modifiers of PP's {picture aiders}

Advantages of Conceptual Dependency

- Using these primitives involves fewer inference rules.
- So, many inference rules already represented in CD structure.
- Moreover, the holes in the initial structure help to focus on the points still to be established.

Disadvantages of Conceptual Dependency

- Knowledge must decompose into fairly low-level primitives.
- Impossible or difficult to find the correct set of primitives.
- Also, 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.
- Moreover, complex representations require a lot of storage.

Scripts

Scripts Strong Slot

- A script is a structure that prescribes a set of circumstances which could be expected to follow on from one another.
- It is similar to a thought sequence or a chain of situations which could be anticipated.
- It could be considered to consist of a number of slots or frames but with more specialized roles.

Scripts are beneficial because:

- Events tend to occur in known runs or patterns.
- Causal relationships between events exist.
- Entry conditions exist which allow an event to take place
- Prerequisites exist for events taking place. E.g. when a student progresses through a degree scheme or when a purchaser buys a house.

Script Components

Each script contains the following main components.

- **Entry Conditions:** Must be satisfied before events in the script can occur.
- **Results:** Conditions that will be true after events in script occur.
- **Props:** Slots representing objects involved in the events.
- **Roles:** Persons involved in the events.
- **Track:** the specific variation on the more general pattern in the script. Different tracks may share many components of the same script but not all.
- **Scenes:** The sequence of events that occur. Events represented in conceptual dependency form.

Advantages and Disadvantages of Script

Advantages

- Capable of predicting implicit events
- Single coherent interpretation may be build up from a collection of observations.

Disadvantage

- More specific (inflexible) and less general than frames.
- Not suitable to represent all kinds of knowledge.

To deal with inflexibility, smaller modules called memory organization packets (MOP) can combine in a way that appropriates for the situation.

Script Example

- It must activate based on its significance.
- If the topic important, then the script should open.
- If a topic just mentioned, and then a pointer to that script could hold.
- For example, given “John enjoyed the play in theatre”, a script “Play in Theatre” suggested above invoke.
- All implicit questions can answer correctly.

Here the significance of this script is high.

- Did John go to the theatre?
- Also, Did he buy the ticket?
- Did he have money?

If we have a sentence like “John went to the theatre to pick his daughter”, then invoking this script will lead to many wrong answers.

- Here significance of the script theatre is less.

Script : Play in theater	Various Scenes
Track: Play in Theater Props: <ul style="list-style-type: none"> • Tickets • Seat • Play Roles: <ul style="list-style-type: none"> • Person (who wants to see a play) – P • Ticket distributor – TD • Ticket checker – TC Entry Conditions: <ul style="list-style-type: none"> • P wants to see a play • P has a money Results: <ul style="list-style-type: none"> • P saw a play • P has less money • P is happy (optional if he liked the play) 	Scene 1: Going to theater <ul style="list-style-type: none"> • P PTRANS P into theater • P ATTEND eyes to ticket counter
	Scene 2: Buying ticket <ul style="list-style-type: none"> • P PTRANS P to ticket counter • P MTRANS (need a ticket) to TD • TD ATRANS ticket to P
	Scene 3: Going inside hall of theater and sitting on a seat <ul style="list-style-type: none"> • P PTRANS P into Hall of theater • TC ATTEND eyes on ticket POSS_by P • TC MTRANS (showed seat) to P • P PTRANS P to seat • P MOVES P to sitting position
	Scene 4: Watching a play <ul style="list-style-type: none"> • P ATTEND eyes on play • P MBUILD (good moments) from play
	Scene 5: Exiting <ul style="list-style-type: none"> • P PTRANS P out of Hall and theater

Getting significance from the story is not straightforward. However, some heuristics can apply to get the value.

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 articles in the Encyclopaedia Britannica.
- Also, Both Implicit and Explicit knowledge encoded.
- Moreover, Emphasis on study of underlying information (assumed by the authors but not needed to tell to the readers).

Example: Suppose we read that Wellington learned of Napoleon's death

- Then we (humans) can conclude Napoleon never knew that Wellington had died.

How do we do this?

So, We require special implicit knowledge or commonsense such as:

- We only die once.
- You stay dead.
- Moreover, You cannot learn anything when dead.
- Time cannot go backward.

Why build large knowledge bases:

1. 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.

2. Form and Content

- Moreover, Knowledge representation may not be suitable for AI. Commonsense strategies could point out where difficulties in content may affect the form.

3. Shared Knowledge

- Also, Should allow greater communication among systems with common bases and assumptions.

How is CYC coded?

- By hand.
- Special CYCL language:
- LISP-like.
- Frame-based
- Multiple inheritances
- Slots are fully fledged objects.
- Generalized inheritance — any link not just *isa* and *instance*.

D.N.R.COLLEGE(AUTONOMOUS):BHIMAVARAM

M.C.A DEPARTMENT



ARTIFICIAL INTELLIGENCE

Presented by

L.SOWJANYA

UNIT IV

Expert System, Concepts and Characteristics, Applications and Domains of Expert System, Elements Of an Expert System, Stages in the Development of an Expert System, Semantic Nets, Frames Speech Recognition, Forms of Learning, Inductive Learning, Learning Decision Trees, Single Layer Feed Forward, Multi Layer Feed Forward Neural Networks.

UNIT-4

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 complex 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.

Examples: There are many examples of expert system.

- **MYCIN:** One of the earliest expert systems based on backward chaining. It can identify various bacteria that can cause severe infections and can also recommend drugs based on the person's weight.
- **DENDRAL:** It was an artificial intelligence based expert system used for chemical analysis. It used a substance's spectrographic data to predict its molecular structure.
- **R1/XCON:** It could select specific software to generate a computer system wished by the user.
- **PXDES:** It could easily determine the type and the degree of lung cancer in a patient based on the data.
- **CaDet:** It is a clinical support system that could identify cancer in its early stages in patients.
- **DXplain:** It was also a clinical support system that could suggest a variety of diseases based on the findings of the doctor.

Characteristics of an 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.

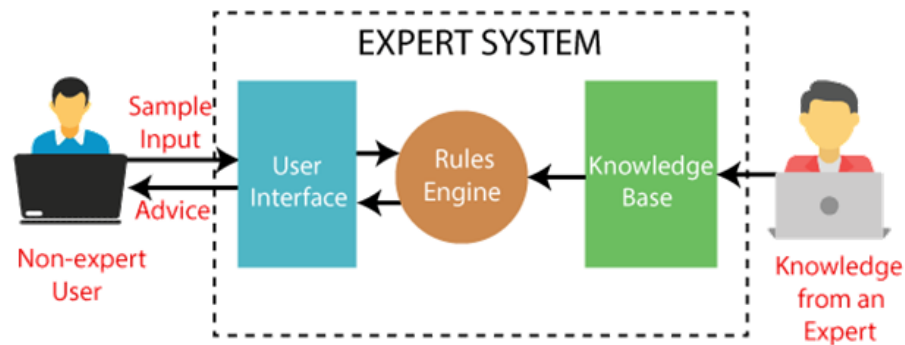
Applications and Domains of Expert System

The following table shows where ES can be applied.

Application	Description
Design Domain	Camera lens design, automobile design.
Medical Domain	Diagnosis Systems to deduce cause of disease from observed data, conduction medical operations on humans.
Monitoring Systems	Comparing data continuously with observed system or with prescribed behavior such as leakage monitoring in long petroleum pipeline.

Process Control Systems	Controlling a physical process based on monitoring.
Knowledge Domain	Finding out faults in vehicles, computers.
Finance/Commerce	Detection of possible fraud, suspicious transactions, stock market trading, Airline scheduling, cargo scheduling.

Components of an expert system:



- **Knowledge base:** The knowledge base represents facts and rules. It consists of knowledge in a particular domain as well as rules to solve a problem, procedures and intrinsic data relevant to the domain.

- **Inference engine:** The function of the inference engine is to fetch the relevant knowledge from the knowledge base, interpret it and to find a solution relevant to the user's problem. The inference engine acquires the rules from its knowledge base and applies them to the known facts to infer new facts. Inference engines can also include an explanation and debugging abilities.
- **Knowledge acquisition and learning module:** The function of this component is to allow the expert system to acquire more and more knowledge from various sources and store it in the knowledge base.
- **User interface:** This module makes it possible for a non-expert user to interact with the expert system and find a solution to the problem.
- **Explanation module:** This module helps the expert system to give the user an explanation about how the expert system reached a particular conclusion.

Development of Expert Systems: General Steps

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

Identify Problem Domain

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

Design the System

- Identify the ES Technology
- Know and establish the degree of integration with the other systems and databases.
- Realize how the concepts can represent the domain knowledge best.

Develop the Prototype

From Knowledge Base: The knowledge engineer works to –

- Acquire domain knowledge from the expert.

- Represent it in the form of If-THEN-ELSE rules.

Test and Refine the Prototype

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

Develop and Complete the ES

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

Maintain the System

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

Advantages:

- Low accessibility cost.
- Fast response.
- Not affected by emotions unlike humans.
- Low error rate.
- Capable of explaining how they reached a solution.

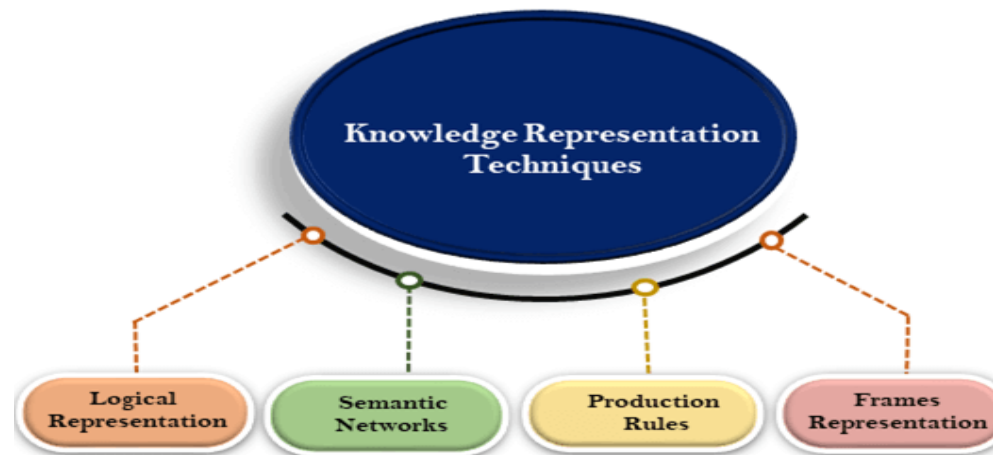
Disadvantages:

- Expert system have no emotions.
- Common sense is the main issue of the expert system.
- It is developed for a specific domain.
- It needs to be updated manually. It does not learn itself.
- Not capable to explain the logic behind the decision.

Techniques of knowledge representation

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

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



2. Semantic Network Representation

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

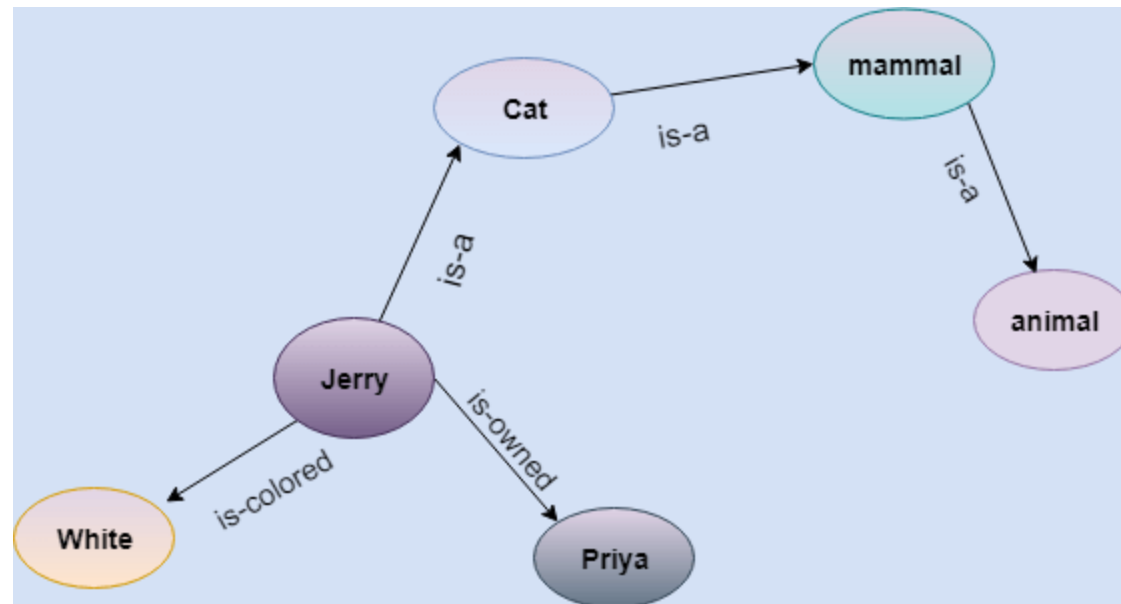
This representation consists of mainly two types of relations:

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

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

Statements:

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



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

Drawbacks in Semantic representation:

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

Advantages of Semantic network:

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

3. Frame Representation

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

Facets: The various aspects of a slot is known as **Facets**. Facets are features of frames which enable us to put constraints on the frames. Example: IF-NEEDED facts are called when data of any particular slot is needed. A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values. A frame is also known as **slot-filter knowledge representation** in artificial intelligence.

Frames are derived from semantic networks and later evolved into our modern-day classes and objects. A single frame is not much useful. Frames system consist of a collection of frames which are connected. In the frame, knowledge about an object or event can be

stored together in the knowledge base. The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

Example: 1

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

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

Advantages of frame representation:

1. The frame knowledge representation makes the programming easier by grouping the related data.
2. The frame representation is comparably flexible and used by many applications in AI.

3. It is very easy to add slots for new attribute and relations.
4. It is easy to include default data and to search for missing values.
5. Frame representation is easy to understand and visualize.

Disadvantages of frame representation:

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

Subsets of AI: Following are the most common subsets of AI:

- Machine Learning
- Deep Learning
- Natural Language processing
- Expert System
- Robotics
- Machine Vision
- Speech Recognition

Speech Recognition:

Speech recognition is a technology which enables a machine to understand the spoken language and translate into a machine-readable format. It can also be said as automatic Speech recognition and computer speech recognition. **It is a way to talk with a computer, and on the basis of that command, a computer can perform a specific task.**

There is some speech recognition software which has a limited vocabulary of words and phrase. This software requires unambiguous spoken language to understand and perform specific task. Today's there are various software or devices which contains speech recognition technology such as Cortana, Google virtual assistant, Apple Siri, etc.

We need to train our speech recognition system to understand our language. In previous days, these systems were only designed to convert the speech to text, but now there are various devices which can directly convert speech into commands.

Speech recognition systems can be used in the following areas:

- **System control or navigation system**
- **Industrial application**
- **Voice dialing system**

There are two types of speech recognition

1. **Speaker Dependent**
2. **Speaker Independent**

FORMS OF learning

1. **Supervised learning**
2. **Unsupervised learning**
3. **Reinforcement learning**

1) Supervised Learning

Supervised learning is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it, and on that basis, it predicts the output.

The system creates a model using labeled data to understand the datasets and learn about each data, once the training and processing are done then we test the model by providing a sample data to check whether it is predicting the exact output or not.

The goal of supervised learning is to map input data with the output data. The supervised learning is based on supervision, and it is the same as when a student learns things in the supervision of the teacher. The example of supervised learning is **spam filtering**.

Supervised learning can be grouped further in two categories of algorithms:

- **Classification**
- **Regression**

2) Unsupervised Learning

Unsupervised learning is a learning method in which a machine learns without any supervision.

The training is provided to the machine with the set of data that has not been labeled, classified, or categorized, and the algorithm needs to act on that data without any supervision. The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.

In unsupervised learning, we don't have a predetermined result. The machine tries to find useful insights from the huge amount of data. It can be further classified into two categories of algorithms:

- **Clustering**
- **Association**

3) Reinforcement Learning

Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the environment and explores it. The goal of an agent is to get the most reward points, and hence, it improves its performance.

The robotic dog, which automatically learns the movement of his arms, is an example of Reinforcement learning.

Inductive Learning Algorithm

Inductive Learning Algorithm (ILA) is an iterative and inductive machine learning algorithm which is used for generating a set of a classification rule, which produces rules of the form “IF-THEN”, for a set of examples, producing rules at each iteration and appending to the set of rules.

Basic Idea:

There are basically two methods for knowledge extraction firstly from domain experts and then with machine learning. For a very large amount of data, the domain experts are not very useful and reliable. So we move towards the machine learning approach for this work.

To use machine learning One method is to replicate the experts logic in the form of algorithms but this work is very tedious, time taking and expensive.

So we move towards the inductive algorithms which itself generate the strategy for performing a task and need not instruct separately at each step.

A Decision Tree is an algorithm used for supervised learning problems such as classification or regression. A decision tree or a classification tree is a tree in which each internal (nonleaf) node is labeled with an input feature. The arcs coming from a node labeled with a feature are labeled with each of the possible values of the feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes.

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called **recursive partitioning**. The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions. This process of top-down induction of decision trees is an example of a greedy algorithm, and it is the most common strategy for learning decision trees.

Decision trees used in data mining are of two main types –

- **Classification tree** – when the response is a nominal variable, for example if an email is spam or not.
- **Regression tree** – when the predicted outcome can be considered a real number (e.g. the salary of a worker).

Decision trees are a simple method, and as such has some problems. One of this issues is the high variance in the resulting models that decision trees produce. In order to alleviate this problem, ensemble methods of decision trees were developed. There are two groups of ensemble methods currently used extensively –

- **Bagging decision trees** – These trees are used to build multiple decision trees by repeatedly resampling training data with replacement, and voting the trees for a consensus prediction. This algorithm has been called random forest.
- **Boosting decision trees** – Gradient boosting combines weak learners; in this case, decision trees into a single strong learner, in an iterative fashion. It fits a weak tree to the data and iteratively keeps fitting weak learners in order to correct the error of the previous model.

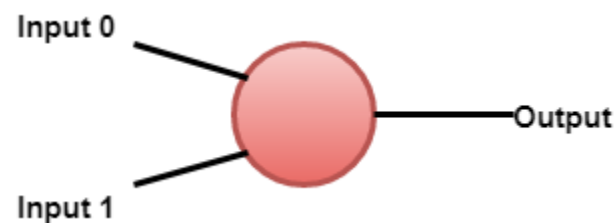
Neural Network

- A Neural Network is a system designed to operate like a human brain. Human information processing takes place through the interaction of many billions of neurons connected to each other sending signals to other neurons.
- Similarly, a Neural Network is a network of artificial neurons, as found in human brains, for solving artificial intelligence problems such as image identification. They may be a physical device or mathematical constructs.
- In other words, Artificial Neural Network is a parallel computational system consisting of many simple processing elements connected to perform a particular task.

For the purpose, an artificial brain was designed is known as a neural network. The neural network is made up many **perceptrons**.

Perceptron is a single layer neural network. It is a binary classifier and part of supervised learning. A simple model of the biological neuron in an artificial neural network is known as the perceptron.

The artificial neuron has input and output.



Representation of perceptron model mathematically.

$$\sum_{i=0}^n w_i x_i + b$$

Human brain has neurons for passing information, similarly neural network has nodes to perform the same task. Nodes are the mathematical functions.

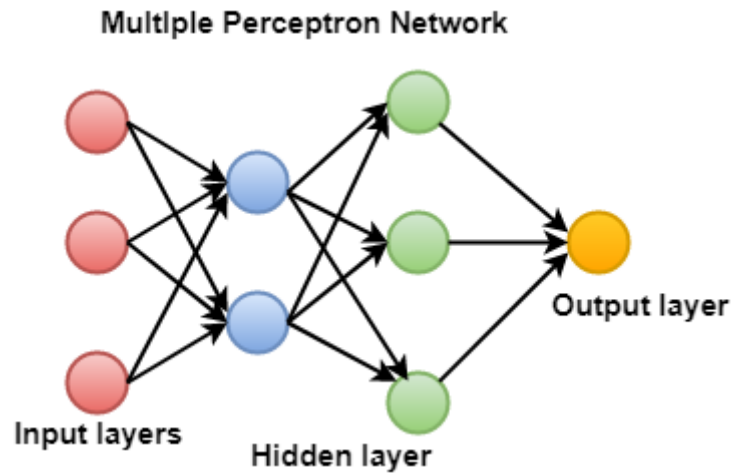
A neural network is based on the structure and function of biological neural networks. A neural network itself changes or learn based on input and output. The information flows through the system affect the structure of the artificial neural network because of its learning and improving the property.

A **Neural Network** is also defined as:

A computing system made of several simple, highly interconnected processing elements, which process information by its dynamic state response to external inputs.

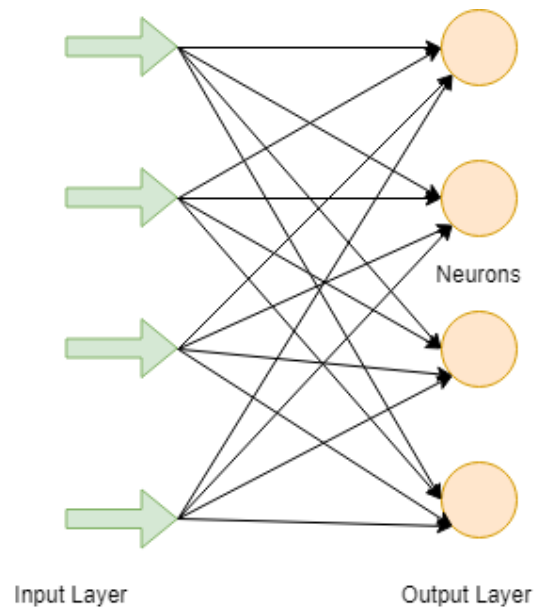
A neural network can be made with multiple perceptrons. Where there are three layers-

- **Input layer:** Input layers are the real value from the data.
- **Hidden layer:** Hidden layers are between input and output layers where three or more layers are deep network.
- **Output layer:** It is the final estimate of the output.



Feedforward Neural Network (Artificial Neuron)

FNN is the purest form of ANN in which input and data travel in only one direction. Data flows in an only forward direction; that's why it is known as **the Feedforward Neural Network**. The data passes through input nodes and exit from the output nodes. The nodes are not connected cyclically. It doesn't need to have a hidden layer. In FNN, there doesn't need to be multiple layers. It may have a single layer also.



It has a front propagate wave that is achieved by using a classifying activation function. All other types of neural network use backpropagation, but FNN can't. In FNN, the sum of the product's input and weight are calculated, and then it is fed to the output. Technologies such as **face recognition** and **computer vision** are used FNN.

Multilayer Perceptron

A **Multilayer Perceptron** has three or more layer. The data that cannot be separated linearly is classified with the help of this network. This network is a fully connected network that means every single node is connected with all other nodes that are in the next layer. A **Nonlinear Activation Function** is used **in Multilayer Perceptron**. It's input and output layer nodes are connected as a directed graph. It is a deep learning method so that for training the network it uses **backpropagation**. It is extensively applied in speech recognition and machine translation technologies.

