APPENDIX-V

(CT)

INTRODUCTION TO ARTIFICIAL NEURAL NETWORK

Q.1. Explain in brief about Artificial Neural Network.

OR What are the characteristics of Artificial Neural Network?

Ans. Characteristics of Artificial Neural Network:

(i) It is neurally implemented mathematical model.

(ii) It contains huge number of interconnected processing elements called neurons to do all operations.

(iii) Information stored in the neurons are basically the weighted linkage of neurons.

(iv) The input signals arrive at the processing elements through connections and connecting weights.

(v) It has the ability to learn, recall and generalize from the given data by suitable assignment and adjustment of weights.

(vi) The collective behavior of the neurons describes its computational power and no single neuron carries specific information.

Q.2. How Simple Neuron Works?

Ans. Working of Simple Neuron:

- Let there are two neurons X and Y which is transmitting signal to another neuron Z. Then, X and Y are input neurons for transmitting signals and Z is output neuron for receiving signal.
- The input neurons are connected to the output neuron, over a interconnection links (A and B) as shown in figure given below.

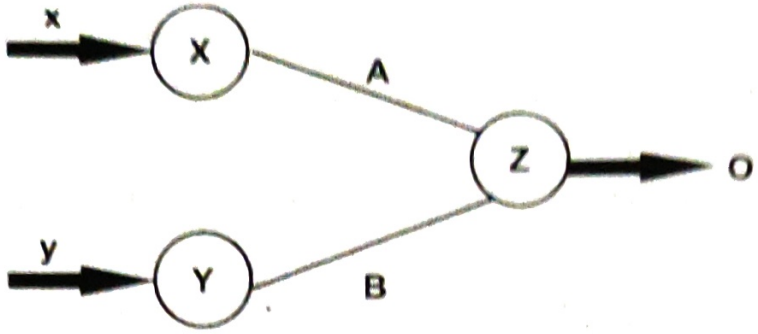


Fig. Architecture of a Simple Artificial Neuron Net

- For above Neuron Architecture, the net input has to be calculated in the way.

  $I = xA + yB$

  where x and y are the activations of the input neurons X and Y. The output z of the output neuron Z can be obtained by applying activations over the net input.

  $O = f(I)$

  Output = Function (net input calculated)

- The function to be applied over the net input is called activation function. There are various activation function possible for this.

Q.3. Write the application of Neural Network.

Ans. Application of Neural Network:

(i) Every new technology need assistance from the previous one i.e. data from previous ones and these data are analyzed so that every pros and cons should be studied correctly. All of these things are possible only through the help of neural network.

(ii) Neural network is suitable for the research on Animal behavior, predator/prey relationships and population cycles.

(iii) It would be easier to do proper valuation of property, buildings, automobiles, machinery etc. with the help of neural network.

(iv) Neural Network can be used in betting on horse races, sporting events, and most importantly in stock market.

(v) It can be used to predict the correct judgment for any crime by using a large data of crime details as input and the resulting sentences as output.

(vi) By analyzing data and determining which of the data has any fault (files diverging from peers) called as Data mining, cleaning and validation can be achieved through neural network.

(vii) Neural Network can be used to predict targets with the help of echo patterns we get from sonar, radar, seismic and magnetic instruments.

(viii) It can be used efficiently in employee hiring so that any company can hire the right employee depending upon the skills the employee has and what should be its productivity in future.

(ix) It has a large application in Medical Research.

(x) It can be used to for Fraud Detection regarding credit cards, insurance or taxes by analyzing the past records.

GENETIC ALGORITHM

Q.4. What is Genetic Algorithm? Explain with example.

OR Describe the foundation of Genetic Algorithm.

Ans. Genetic Algorithm:

- Genetic Algorithms (GAs) are adaptive heuristic Search Algorithms that belong to the larger part of evolutionary algorithms. Genetic Algorithms are based on the ideas of natural selection and genetics.
- These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

- Genetic Algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation.
- In simple words, they simulate "Survival of the Fittest" among individual of consecutive generation for solving a problem. Each generation consist of a population of individuals and each individual represents a point in search space and possible solution.
- Each individual is represented as a string of character/ integer/ float/bits. This string is analogous to the Chromosome.

Foundation of Genetic Algorithms:

Genetic Algorithms are based on an analogy with genetic structure and behaviour of chromosomes of the population. Following is the foundation of GAs based on this analogy:

(i) Individual in population compete for resources and mate.

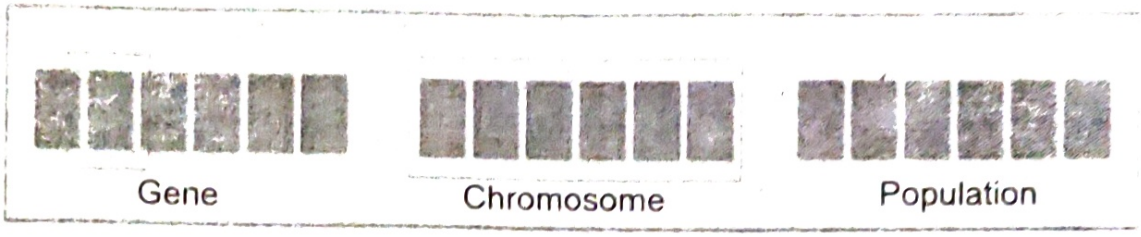
(ii) Those individuals who are successful (fittest) then mate to create more offspring than others.

(iii) Genes from "fittest" parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.

(iv) Thus, each successive generation is more suited for their environment.

(a) Search Space:

- The population of individuals are maintained within search space. Each individual represents a solution in search space for given problem.
- Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes.
- Thus, a chromosome (individual) is composed of several genes (variable components).



(b) Fitness Score:

- A Fitness Score is given to each individual which shows the ability of an individual to "Compete". The individual having optimal fitness score (or near optimal) are sought.
- The GAs maintains the population of individuals (chromosome/solutions) along with their fitness scores. The individuals having better fitness scores are given more chance to reproduce than others. The individuals with better fitness scores are selected who mate and produce better offspring by combining chromosomes of parents.
- The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating new generation when all the mating opportunity of the old population is exhausted. It is hoped that over successive generations better solutions will arrive while least fit die.
- Each new generation has on average more "better genes" than the individual (solution) of previous generations.
- Thus, each new generations have better "Partial Solutions" than previous generations. Once the offspring produced having no significant difference from offspring produced by previous populations, the population is converged. The algorithm is said to be converged to a set of solutions for the problem.

(c) Operators of Genetic Algorithms:

Once the initial generation is created, the algorithm evolves the generation using following operators:

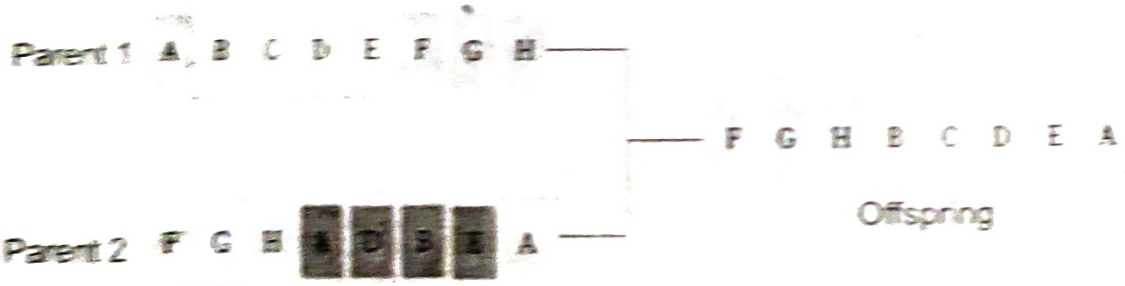
(1) Selection Operator:

The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations.

(2) Crossover Operator:

This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chossen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring).

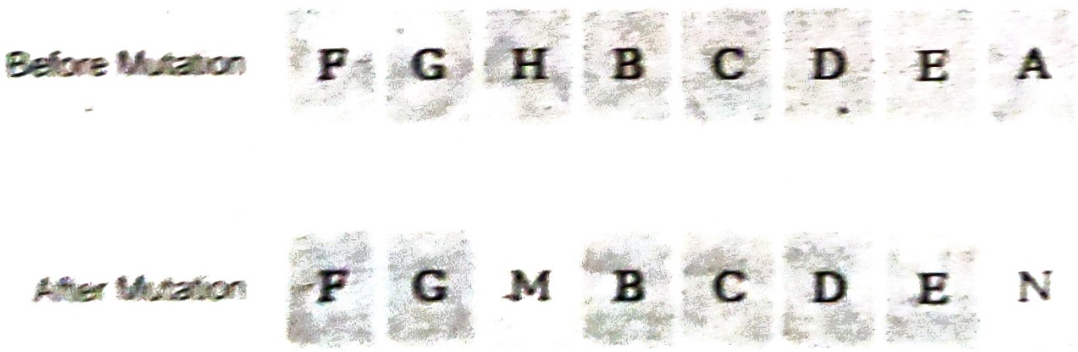
For example:



(3) Mutation Operator:

The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence.

For example:



The whole Algorithm can be summarized as:

(i) Randomly initialize populations

(ii) Determine fitness of population

(iii) Until convergence repeat:

(a) Select parents from population

(b) Crossover and generate new population

(c) Perform mutation on new population

(d) Calculate fitness for new population

Problem and Solution using Genetic Algorithms:

Given a target string, the goal is to produce target string starting from a random string of the same length. In the following implementation, following analogies are made:

- Characters A-Z, a-z, 0-9 and other special symbols are considered as genes
- A string generated by these characters is considered as chromosome/solution/Individual

(CSE)

INTRODUCTION TO INTELLIGENT AGENTS

Q.5. What is an Agent? Give its types.

Ans. Agent:

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

Types of Agents:

(i) Human-Agent:

A human agent has eyes, ears and other organs which work for sensors and hand, legs, vocal tract work for actuators.

(ii) Robotic Agent:

A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.

(iii) Software Agent:

- Software agent can have keystrokes, file contents as sensory input and act on those inputs and display output on the screen.
- Hence, the world around us is full of agents such as thermostat, cellphone, camera and even we are also agents.
- Before moving forward, we should first know about sensors, effectors and actuators.

(a) Sensor:

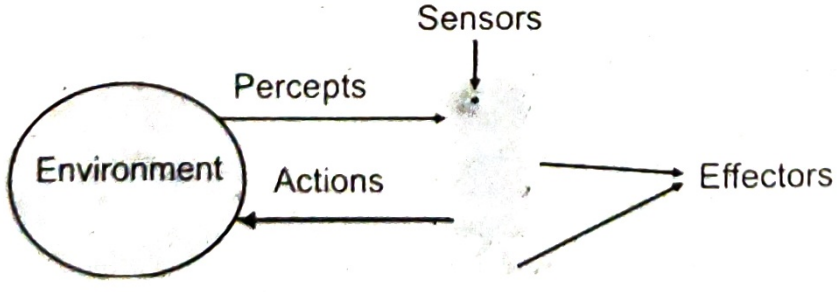
Sensor is a device which detects the change in the environment and sends the information to other electronic devices. An agent observes its environment through sensors.

(b) 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.

(c) Effectors:

Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers, wings, fins and display screen.



Q.6. What is Intelligent Agents? Define the four rules of Agents.

Ans. Intelligent Agents:

An intelligent agent is an autonomous entity which act upon an environment using Sensors and Actuators for achieving goals. An intelligent agent may learn from the environment to achieve their goals. A thermostat is an example of an Intelligent Agent.

Following are the main four Rules for an Intelligent Agents:

(i) **Rule 1:**

An AI agent must have the ability to perceive the environment.

(ii) **Rule 2:**

The observation must be used to make decisions.

(iii) **Rule 3:**

Decision should result in an action.

(iv) **Rule 4:**

The action taken by an AI agent must be a rational action.

RATIONAL AGENT AND THEIR STRUCTURE

Q.7. Explain Rational Agent with structure and define Rationality of Agent.

Ans. Rational Agent:

- A rational agent is an agent which has clear preference, models uncertainty and acts in a way to maximize its performance measure with all possible actions.
- A rational agent is said to perform the right things. AI is about creating rational agents to use for game theory and decision theory for various real world scenarios.
- For an AI agent, the rational action is most important because in AI Reinforcement Learning Algorithm, for each best possible action, agent gets the positive reward and for each wrong action, an agent gets a negative reward.

Structure of an AI Agent:

The task of AI is to design an agent program which implements the agent function. The structure of an intelligent agent is a combination of architecture and agent program. It can be viewed as:

Agent = Architecture + Agent program

Following are the main three terms involved in the structure of an AI agent:

(i) **Architecture:**

Architecture is machinery that an AI agent executes on.

(ii) **Agent Function:**

Agent function is used to map a percept to an action.

(iii) **Agent Program:**

Agent program is an implementation of agent function. An agent program executes on the physical architecture to produce function.

Rationality:

The rationality of an agent is measured by its performance measure. Rationality can be judged on the basis of following points:

(i) Performance measure which defines the success criterion.

(ii) Agent prior knowledge of its environment.

(iii) Best possible actions that an agent can perform.

(iv) The sequence of percepts.

Q.8. Explain PEAS Representation.

OR Describe the model on which AI Agent Work.

Ans. 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 Agent's behavior.

PEAS for Self-driving Cars:

Let's suppose a self-driving car then PEAS Representation will be:

(i) **Performance:**

Safety, Time, Legal drive, Comfort.

(ii) **Environment:**

Roads, Other Vehicles, Road signs, Pedestrian.

(iii) **Actuators:**

Steering, Accelerator, Brake, Signal, Horn.

(iv) **Sensors:**

Camera, GPS, Speedometer, Odometer, Accelerometer, Sonar.

Example of Agents with their PEAS representation:

Sr. No.	Agent	Performance measure	Environment	Actuators	Sensors
(1)	Medical Diagnose	Healthy patient Minimized cost	Patient Hospital Staff	Tests Treatments	Keyboard (Entry of Symptoms)
(2)	Vacuum Cleaner	Cleanness Efficiency Battery life Security	Room Table Wood floor Carpet Various obstacles	Wheels Brushes Vacuum Extractor	Camera Dirt detection sensor Cliff sensor Bump Sensor Infrared
(3)	Part - picking Robot	Percentage of parts in correct bins.	Conveyor belt with parts, Bins	Jointed Arms Hand	Camera Joint angle Sensors

TYPES OF AGENTS

Q.9. Explain the various Types of Agents.

OR Explain Reflex Agent.

OR Explain Model-based Agent.

OR Explain Goal Based and Utility Based Agent.

Ans. Types of Agents:

Agents can be grouped into five classes based on their degree of perceived Intelligence and Capability:

(1) Simple Reflex Agents:

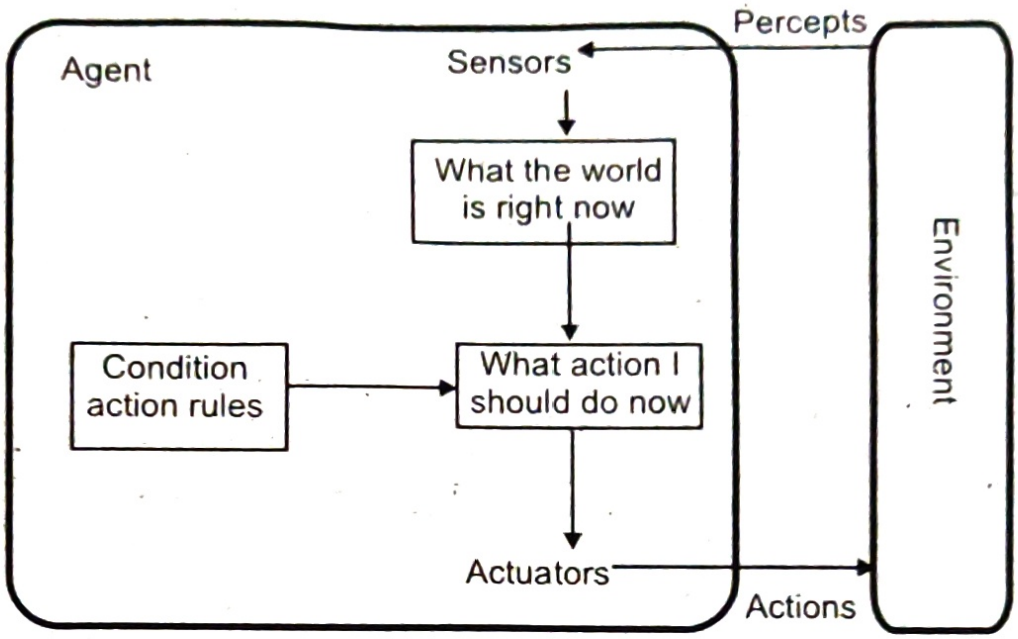
- Simple Reflex Agents ignore the rest of the percept history and act only on the basis of the current percept. Percept history is the history of all that an agent has perceived to date.
- The agent function is based on the condition action rule. A condition action rule is a rule that maps a state i.e. Condition to an action.
- If the condition is true, then the action is taken, else not. This agent function only succeeds when the environment is fully observable.
- For Simple Reflex Agents operating in partially observable environments, infinite loops are often unavoidable.
- It may be possible to escape from infinite loops if the agent can randomize its actions.
- Problems with Simple Reflex Agents are as follows:

(i) Very limited Intelligence.

(ii) No knowledge of Non-perceptual parts of the state.

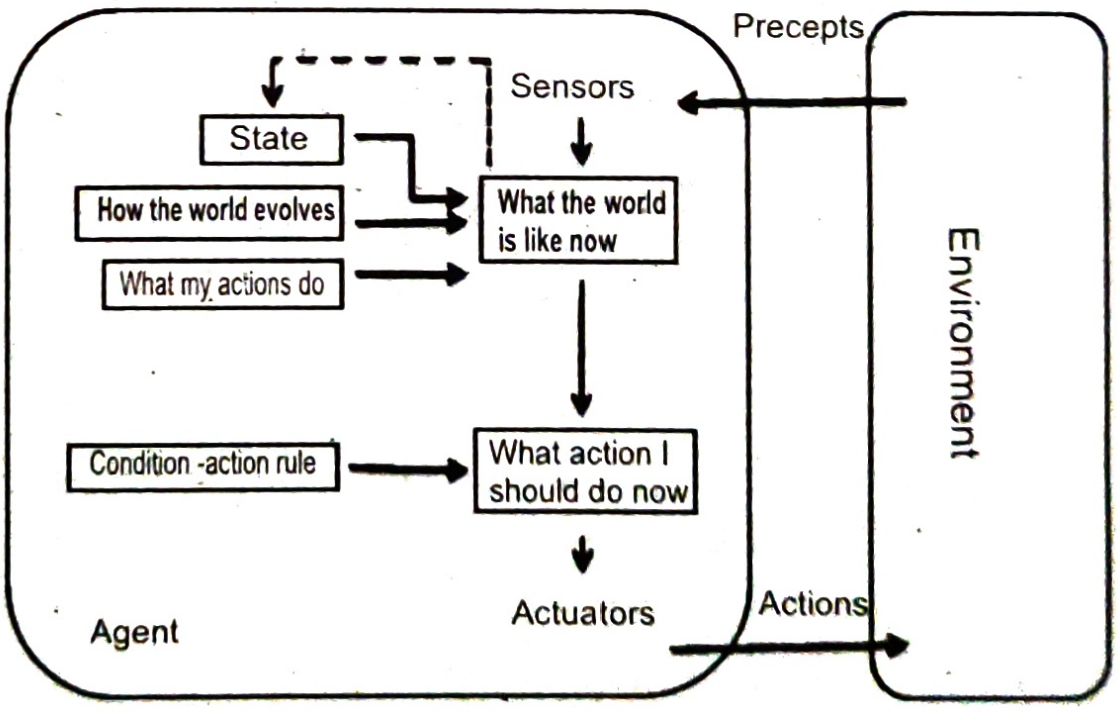
(iii) Usually too big to generate and store.

(iv) If there occurs any change in the environment, then the collection of rules need to be updated.



(2) Model-based Reflex Agents:

- Model-based Reflex Agents works by finding a rule whose condition matches the current situation. A Model-based Agent can handle partially observable environments by the use of a model about the world.
- The agent has to keep track of the internal state which is adjusted by each percept and that depends on the percept history.



- The current state is stored inside the agent which maintains some kind of structure describing the part of the world which cannot be seen.

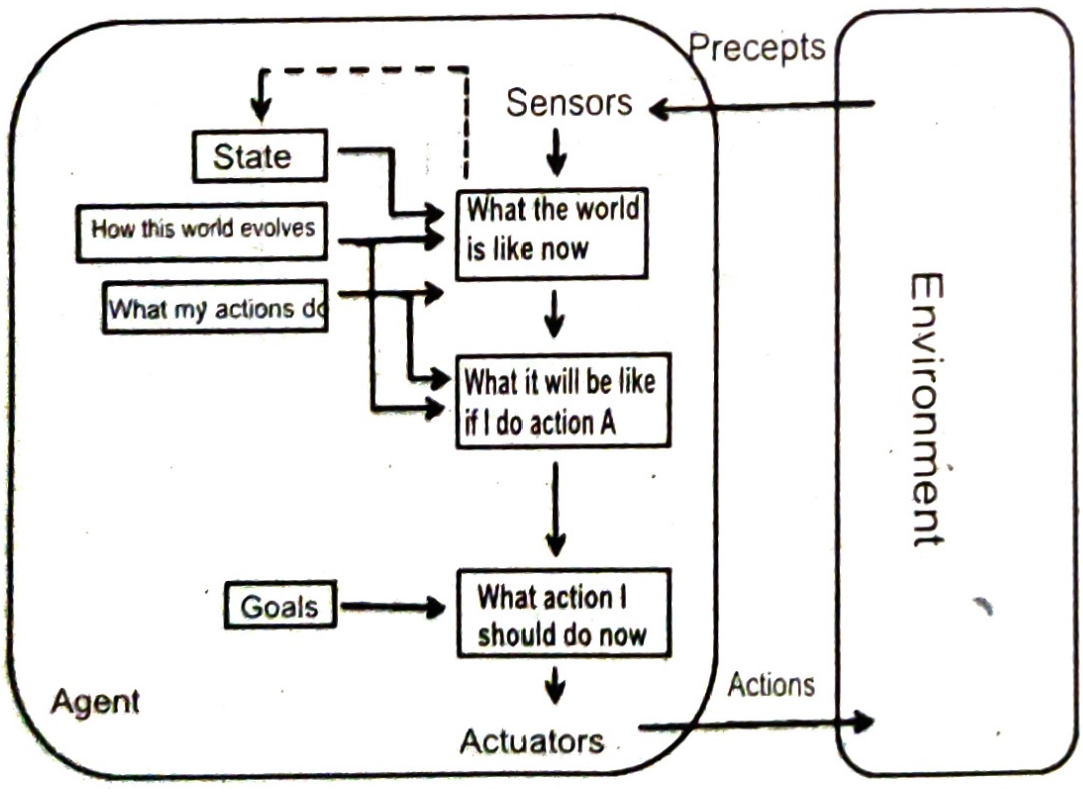
  Updating the state requires information about:

(i) How the world evolves independently from the agent.

(ii) How the agent's actions affect the world.

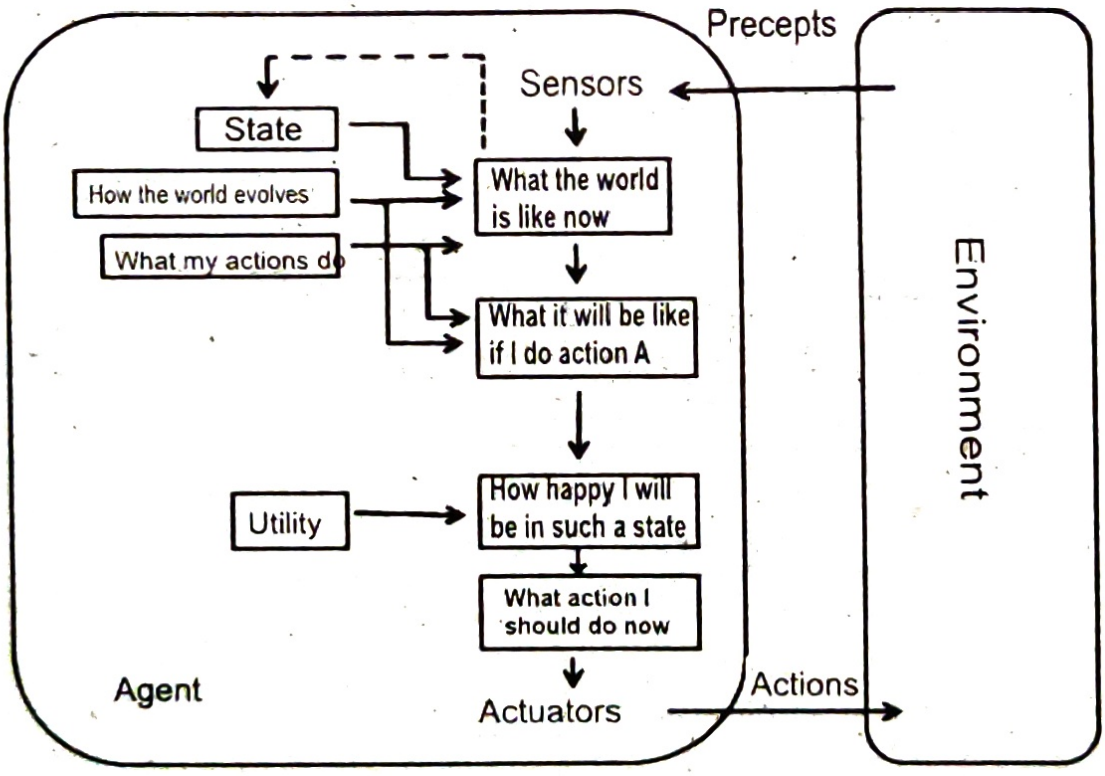
(3) Goal-based Agents:

- Goal-based Agents take decisions based on how far they are currently from their goal (description of desirable situations). Their every action is intended to reduce its distance from the goal.
- This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state. The knowledge that supports its decisions is represented explicitly and can be modified, which makes these agents more flexible.
- They usually require search and planning. The Goal-Based Agent's behavior can easily be changed.



(4) Utility-based Agents:

- The agents which are developed having their end uses as building blocks are called Utility-based Agents. When there are multiple possible alternatives, then to decide which one is best, Utility-based Agents are used.
- They choose actions based on a preference (Utility) for each state. Sometimes achieving the desired goal is not enough. We may look for a quicker, safer, cheaper trip to reach a destination.
- Agent happiness should be taken into consideration. Utility describes how "Happy" the agent is.
- Because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility. A utility function maps a state onto a real number which describes the associated degree of happiness.



(5) Learning Agent:

- A learning agent in Artificial Intelligence is the type of agent that can learn from its past experiences or it has learning

capabilities. It starts to act with basic knowledge and then is able to act and adapt automatically through learning.

A learning agent has mainly four conceptual components, which are as follows:

(i) Learning Element:

Learning Element is responsible for making improvements by learning from the environment.

(ii) Critic:

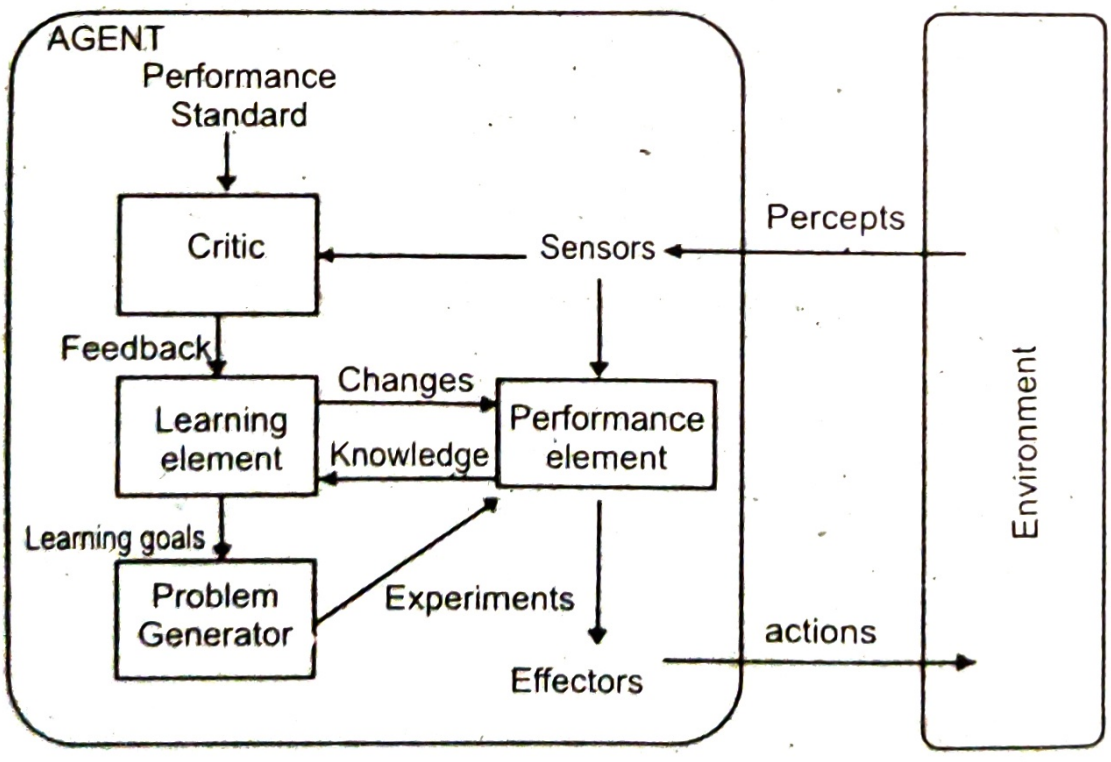
The learning element takes feedback from critics which describes how well the agent is doing with respect to a fixed performance standard.

(iii) Performance Element:

Performance Element is responsible for selecting external action.

(iv) Problem Generator:

Problem Generator is responsible for suggesting actions that will lead to new and informative experiences.



BEHAVIOUR AND ENVIRONMENT OF AGENT

Q.10. What should be behaviour and Environment of Agent?

Ans. Environment of Agent:

- An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself. An environment can be described as a situation in which an agent is present.
- The environment is where agent lives, operate and provide the agent with something to sense and act upon it. An environment is mostly said to be Non-feministic.

Features of Environment:

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

(i) 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.

(ii) 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.

(iii) 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.

(iv) 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.

(v) 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.

(vi) 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.

(vii) 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.

(viii) 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.

(CE)

FUTURE OF ARTIFICIAL INTELLIGENCE

Q.11.What is present scenario of Artificial Intelligence?

Ans. Artificial Intelligence (AI) at Present:

- Before going deep dive into AI in future, first, let's understand what is Artificial Intelligence and at what stage it is at present.
- We can define AI as, "Artificial Intelligence is the ability of machines or computer controlled robot to perform task that

- However, apart from these beneficial uses, one great challenge of AI in healthcare is to ensure its adoption in daily clinical practices.

(b) Cyber Security:

- Undoubtedly, Cyber Security is a priority of each organization to ensure data security. There are some predictions that Cyber Security with AI will have below changes:

(i) With AI tools, security incidents will be monitored.

(ii) Identification of the origin of cyber-attacks with NLP.

(iii) Automation of rule-based tasks and processes with the help of RPA bots.

- However, being a great technology, it can also be used as a threat by attackers. They can use AI in a non-ethical way by using automated attacks that may be intangible to defend.

(c) Transportation:

- The fully autonomous vehicle is not yet developed in the transportation sector, but researchers are reaching in this field.
- AI and machine learning are being applied in the cockpit to help reduce workload, handle pilot stress and fatigue and improve on-time performance.
- There are several challenges to the adoption of AI in transportation, especially in areas of public transportation. There's a great risk of over-dependence on automatic and autonomous systems.

PROBLEM SOLVING METHODS

Q.15. Explain the various Problem Solving Methods.

OR Enlist the components to formulate the Associated Problems.

Ans. Problem Solving Methods:

- The Reflex Agent of AI directly maps states into action. Whenever these agents fail to operate in an environment where the state of mapping is too large and not easily performed by the agent, then the stated problem dissolves and sent to a problem-solving domain which breaks the large stored problem into the smaller storage area and resolves one by one. The final integrated action will be the desired outcomes.
- On the basis of the problem and their working domain, different types of problem solving agent defined and use at an atomic level without any internal state visible with a problem solving algorithm. The problem solving agent performs precisely by defining problems and several solutions. So, we can say that problem solving is part of Artificial Intelligence that encompasses a number of techniques such as a tree, B-tree, Heuristic Algorithms to solve a problem.
- We can also say that a problem-solving agent is a result-driven agent and always focuses on satisfying the goals.

Steps of Problem Solving in AI:

The problem of AI is directly associated with the nature of humans and their activities. So, we need a number of finite steps to solve a problem which makes human easy works.

These are the following steps which require to solve a problem:

(i) Goal Formulation:

Goal Formulation is the first and simple step in problem solving. It organizes finite steps to formulate a target/goals which require some action to achieve the goal. Today the formulation of the goal is based on AI agents.

(ii) Problem Formulation:

Problem Formulation is one of the core steps of problem solving which decides what action should be taken to achieve

the formulated goal. In AI this core part is dependent upon software agent which consisted of the following components to formulate the associated problem.

Components to formulate the Associated Problem:

(a) Initial State:

Initial State requires an initial state for the problem which starts the AI agent towards a specified goal. In this state new methods also initialize problem domain solving by a specific class.

(b) Action:

Action stage of problem formulation works with function with a specific class taken from the initial state and all possible actions done in this stage.

(c) Transition:

Transition stage of problem formulation integrates the actual action done by the previous action stage and collects the final stage to forward it to their next stage.

(d) Goal Test:

Goal Test stage determines that the specified goal achieved by the integrated transition model or not, whenever the goal achieves stop the action and forward into the next stage to determines the cost to achieve the goal.

(e) Path Costing:

Path Costing component of problem solving numerical assigned what will be the cost to achieve the goal. It requires all hardware software and human working cost.

SEARCH STRATEGIES

Q.16. What is Search Algorithms in AI? Classify search Algorithm.

Ans. Search Algorithms in AI:

Artificial Intelligence is the study of building agents that act Rationally. Most of the time, these agents perform some kind

of search algorithm in the background in order to achieve their tasks.

Search Problem Consists of:

(i) **State Space:** Set of all possible states where you can be.

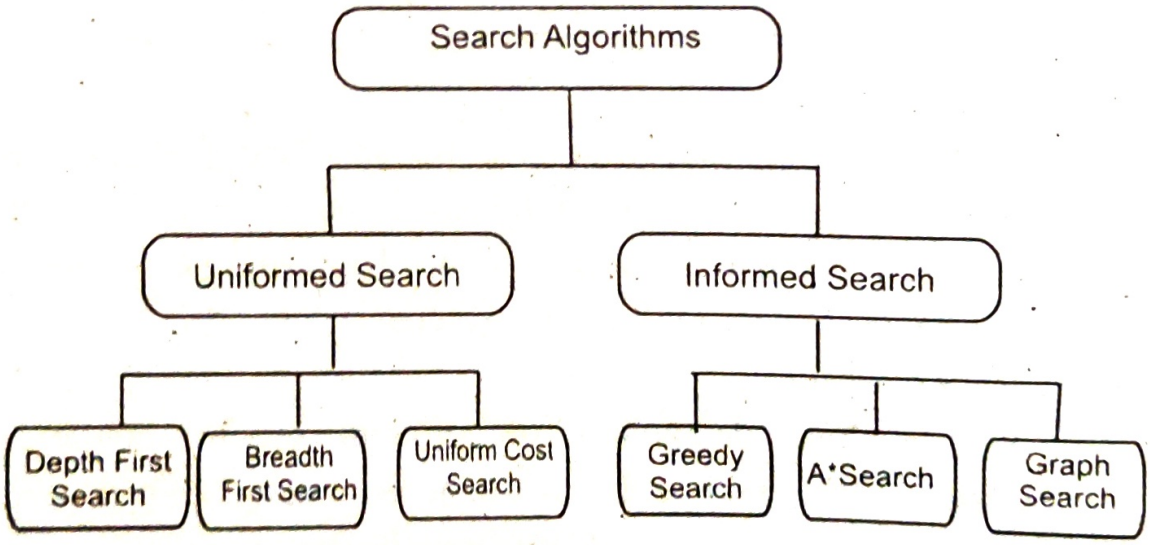
(ii) **Start State:** The state from where the search begins.

(iii) **Goal Test:** A function that looks at the current state returns. Whether or not it is the goal state.

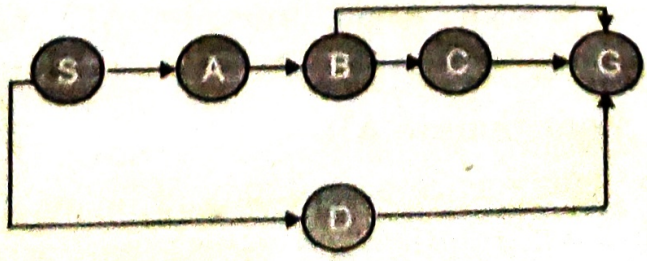
The Solution to a search problem is a sequence of actions, called the plan that transforms the start state to the goal state. This plan is achieved through Search Algorithms.

Types of Search Algorithms:

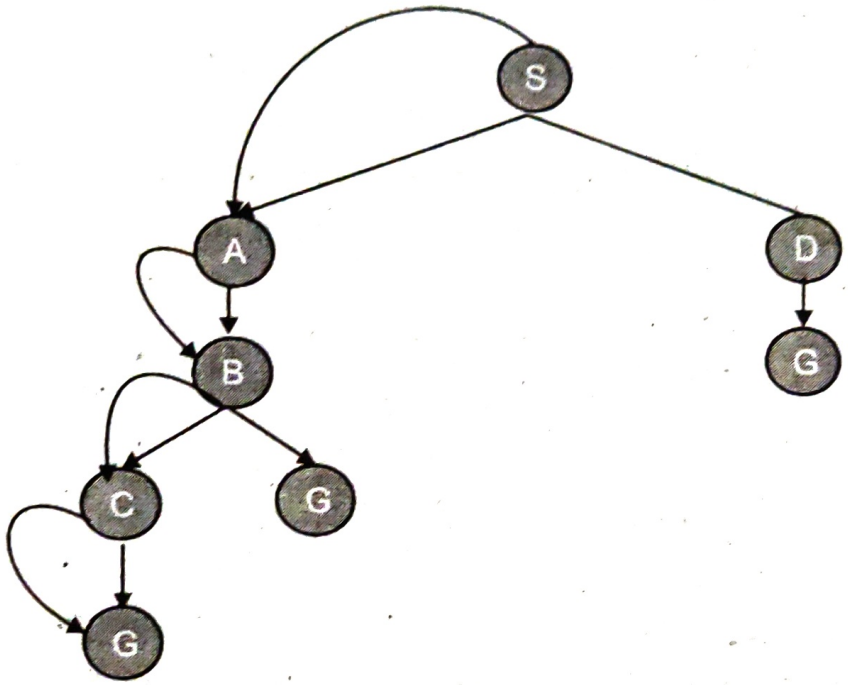
There are far too many powerful Search Algorithms out there to fit in a single article. Instead, we will learn six of the fundamental Search Algorithms which are divided into two categories, as shown below:



Ex.1. Which solution would DFS find to move from node S to node G if run on the graph below?



Soln. The equivalent search tree for the above graph is as follows. As DFS traverses the tree "deepest node first", it would always pick the deeper branch until it reaches the solution (or it runs out of nodes and goes to the next branch). The traversal is shown in blue arrows.



Path: S –> A –> B –> C –> G

= the depth of the search tree = the number of levels of the search tree.

= number of nodes in level.

Time Complexity:

Equivalent to the number of nodes traversed in DFS.

Space Complexity:

Equivalent to how large can the fringe get.

Completeness:

DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.

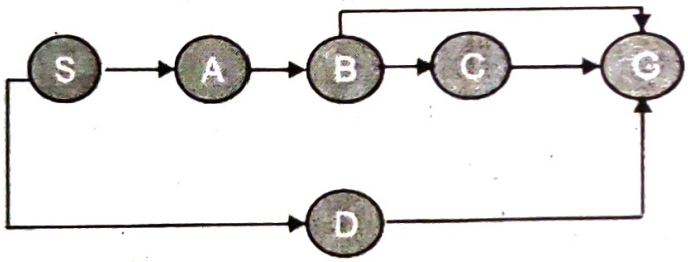
Optimality:

DFS is not optimal, meaning the number of steps in reaching the solution, or the cost spent in reaching it is high.

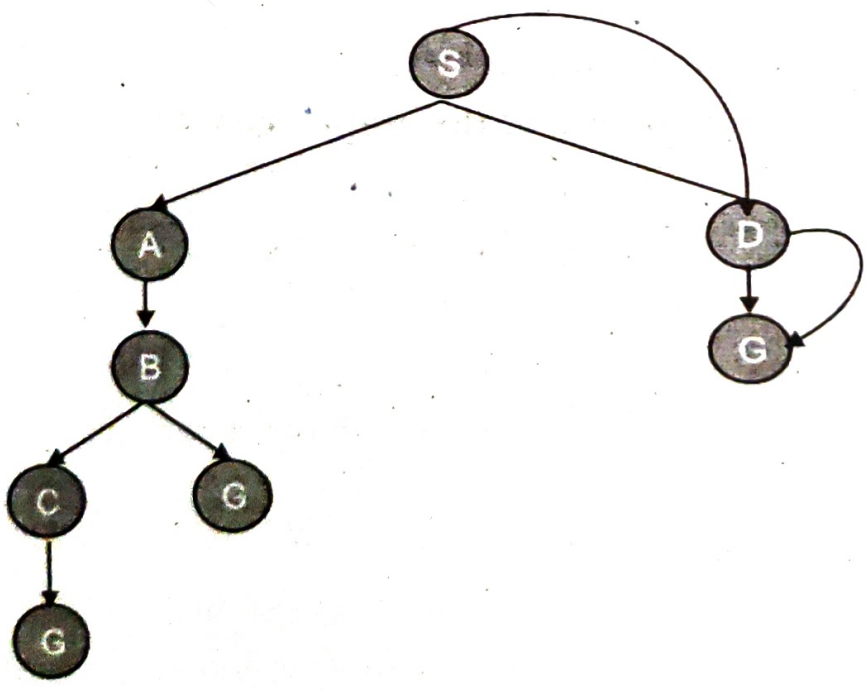
Breadth First Search:

Breadth First Search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'Search Key') and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level. It is implemented using a queue.

Ex.2. Which solution would BFS find to move from node S to node G if run on the graph below?



Soln. The equivalent search tree for the above graph is as follows. As BFS traverses the tree "shallowest node first", it would always pick the shallower branch until it reaches the solution (or it runs out of nodes and goes to the next branch). The traversal is shown in blue arrows.



Path: S – > D –> G

= the depth of the shallowest solution.

= number of nodes in level.

Time Complexity:

Equivalent to the number of nodes traversed in BFS until the shallowest solution.

Space Complexity:

Equivalent to how large can the fringe get.

Completeness:

BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.

Optimality:

BFS is optimal as long as the costs of all edges are equal.

Uniform Cost Search:

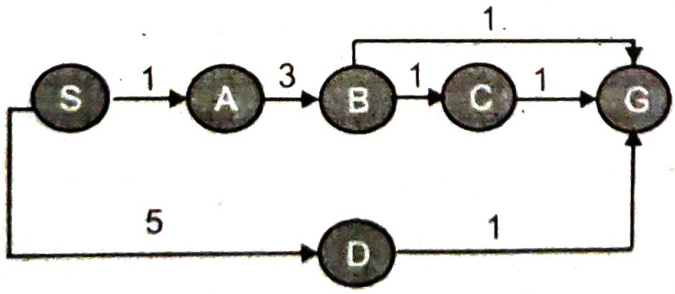
UCS is different from BFS and DFS because here the costs come into play. In other words, traversing via different edges might not have the same cost. The goal is to find a path where the cumulative sum of costs is the least.

Cost of a node is defined as:

cost(node) = cumulative cost of all nodes from root

cost(root) = 0

Ex.3. Which solution would UCS find to move from node S to node G if run on the graph below?



Soln. The equivalent search tree for the above graph is as follows. The cost of each node is the cumulative cost of reaching that node from the root. Based on the UCS strategy, the path with

Q.17. Explain the various Search Strategies.

Ans. Search Strategies:

The algorithms have information on the goal state, which helps in more efficient searching. This information is obtained by something called a Heuristic.

In this section, we will discuss the following Search Algorithms.

(i) Greedy Search

(ii) A* Tree Search

(iii) A* Graph Search

Search Heuristics:

In an informed search, A Heuristic is a function that estimates how close a state is to the goal state.

For example:

Manhattan distance, Euclidean distance, etc. (Lesser the distance, closer the goal.) Different heuristics are used in different informed algorithms discussed below.

Q.18. What is Greedy Search? Explain with example.

Ans. Greedy Search:

In Greedy Search, we expand the node closest to the goal node. The "closeness" is estimated by a heuristic h(x).

Heuristic:

A Heuristic is defined as:

h(x) = Estimate of distance of node x from the goal node.

Lower the value of h(x), closer is the node from the goal.

Strategy:

Expand the node closest to the goal state, i.e. expand the node with a lower Heuristic value.

Ex. Find the path from S to G using greedy search. The heuristic values h of each node below the name of the node.

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

Q.21. Explain the various Local Search Algorithms and Optimization Problem.

Ans. Local Search Algorithms and Optimization Problem:

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

(i) Keeping different paths in the memory and

(ii) 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 matters 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 neighbors of that node generally.

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

(i) Local Search Algorithms use a very little or constant amount of memory as they operate only on a single path.

(ii) Most often, they find a reasonable solution in large or infinite state spaces where the classical or systematic algorithms do not work.

Q.22. Does the Local Search Algorithm work for a pure Optimized Problem?

Ans. Yes, 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. Unfor-

tunately, 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.

Q.23. Explain the working of a Local Search Algorithm.

Ans. Working of a Local Search Algorithm:

Let's understand the working of a Local Search Algorithm with the help of an example:

Consider the below state-space landscape having both:

- **Location:** It is defined by the state.
- **Elevation:** It is defined by the value of the Objective Function or Heuristic Cost Function.

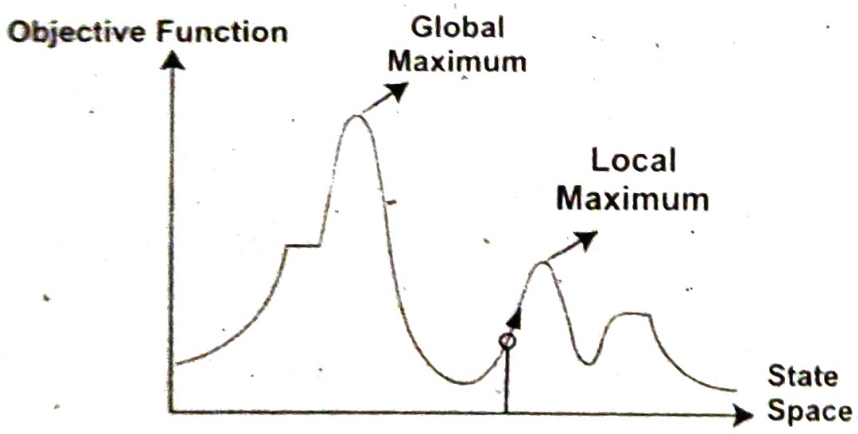


Fig. A one-dimensional state-space landscape in which elevation corresponds to the objective function

The Local Search Algorithm explores the above landscape by finding the following two points:

(i) Global Minimum:

If the elevation corresponds to the cost, then the task is to find the lowest valley, which is known as Global Minimum.

(ii) **Global Maxima:**

If the elevation corresponds to an objective function, then it finds the highest peak which is called as Global Maxima. It is the highest point in the valley.

We will understand the working of these points better in Hill-climbing Search.

Below are some different types of Local Searches:

- Hill-climbing Search
- Simulated Annealing
- Local Beam Search

Hill Climbing Algorithm in AI:

Hill Climbing Search is a local search problem. The purpose of the hill climbing search is to climb a hill and reach the topmost peak/ point of that hill. It is based on the heuristic search technique where the person who is climbing up on the hill estimates the direction which will lead him to the highest peak.

State-space Landscape of Hill Climbing Algorithm:

To understand the concept of Hill Climbing Algorithm, consider the below landscape representing the goal state/ peak and the current state of the climber. The topographical regions shown in the figure can be defined as:

(a) **Global Maximum:**

Global Maximum is the highest point on the hill, which is the goal state.

(b) **Local Maximum:**

Local Maximum is the peak higher than all other peaks but lower than the Global Maximum.

(c) **Flat Local Maximum:**

Flat Local Maximum is the flat area over the hill where it has no uphill or downhill. It is a saturated point of the hill.

(d) Shoulder:

Shoulder is also a flat area where the summit is possible.

(e) Current State:

Current State is the current position of the person.

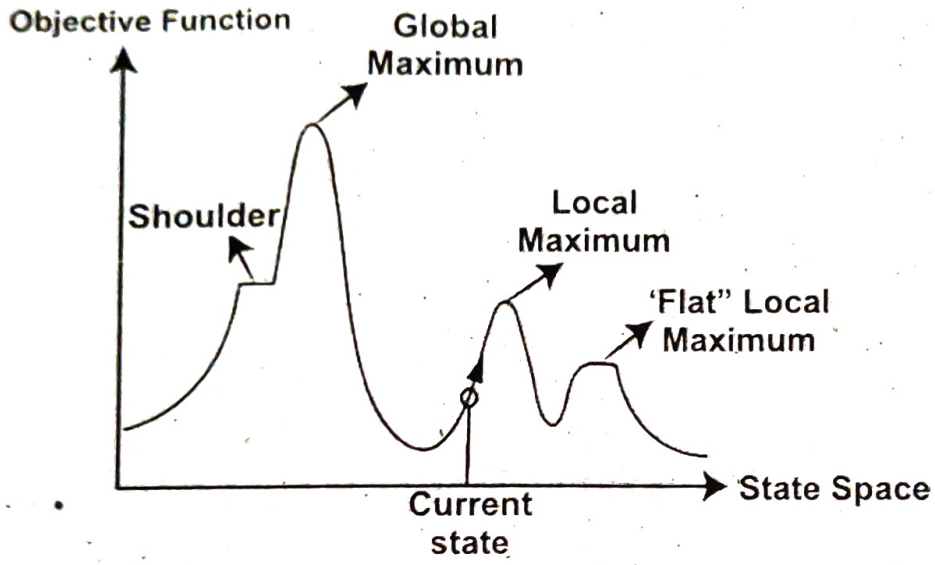
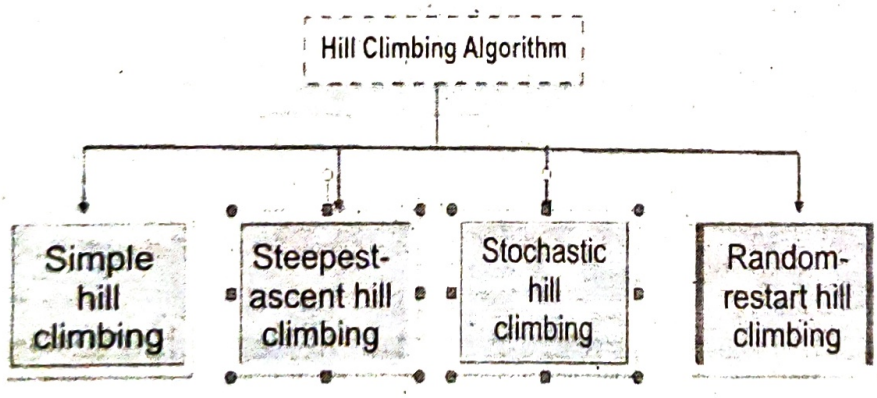


Fig. A one-dimensional state-space landscape in which elevation corresponds to the objective function

Types of Hill Climbing Search Algorithm:

There are following types of Hill Climbing Search:



(1) Simple Hill Climbing Search:

Simple Hill Climbing is the simplest technique to climb a hill. The task is to reach the highest peak of the mountain. Here, the movement of the climber depends on his move/steps. If he finds his next step better than the previous

one, he continues to move else remain in the same state. This search focus only on his previous and next step.

Simple Hill Climbing Algorithm:

(i) Create a CURRENT node, NEIGHBOUR node and a GOAL node.

(ii) If the CURRENT node=GOAL node, return GOAL and terminate the search.

(iii) Else CURRENT node<= NEIGHBOUR node, move ahead.

(iv) Loop until the goal is not reached or a point is not found.

(2) Steepest-ascent Hill Climbing:

Steepest-ascent Hill Climbing is different from simple hill climbing search. Unlike simple hill climbing search, It considers all the successive nodes, compares them and choose the node which is closest to the solution. Steepest Hill Climbing Search is similar to best first search because it focuses on each node instead of one.

Note:

Both simple, as well as Steepest-ascent Hill Climbing Search, fails when there is no closer node.

Steepest-ascent Hill Climbing Algorithms:

(i) Create a CURRENT node and a GOAL node.

(i) If the CURRENT node=GOAL node, return GOAL and terminate the search.

(ii) Loop until a better node is not found to reach the solution.

(iii) If there is any better successor node present, expand it.

(iv) When the GOAL is attained, return GOAL and terminate.

(3) Stochastic Hill Climbing:

Stochastic Hill Climbing does not focus on all the nodes. It selects one node at random and decides whether it should be expanded or search for a better one.

(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.

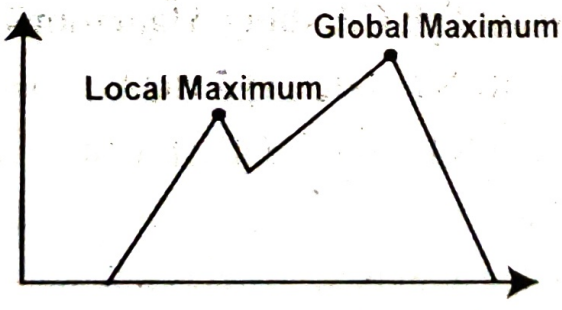
Q.24. What are the limitations of Hill Climbing Algorithm?

Ans. Limitations of Hill Climbing Algorithm:

Hill climbing algorithm is a fast and furious approach. It finds the solution state rapidly because it is quite easy to improve a bad state. But, there are following limitations of this search which are as follows:

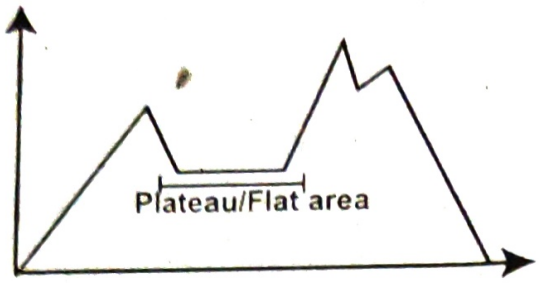
(i) Local Maxima:

Local Maxima is that peak of the mountain which is highest than all its neighboring states but lower than the global maxima. It is not the goal peak because there is another peak higher than it.



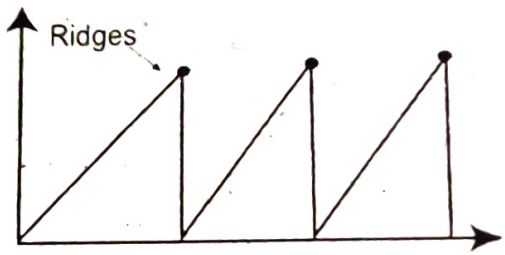
(ii) Plateau:

Plateau is a flat surface area where no uphill exists. It becomes difficult for the climber to decide that in which direction he should move to reach the goal point. Sometimes, the person gets lost in the flat area.



(iii) Ridges:

Ridges is a challenging problem where the person finds two or more local maxima of the same height commonly. It becomes difficult for the person to navigate the right point and stuck to that point itself.



Q.25. What is the alternate option for Hill Climbing Algorithm?

Ans. Alternate to Hill Climbing Algorithm:

(1) Simulated Annealing:

- Simulated Annealing is similar to the Hill Climbing Algorithm. It works on the current situation. It picks a random move instead of picking the best move. If the move leads to the improvement of the current situation, it is always accepted as a step towards the solution state, else it accepts the move having a probability less than 1.
- This search technique was first used in 1980 to solve VLSI layout problems. It is also applied for factory scheduling and other large optimization tasks.

(2) Local Beam Search:

- Local Beam Search is quite different from random-restart search. It keeps track of k states instead of just one. It selects

k randomly generated states and expand them at each step. If any state is a goal state, the search stops with success. Else it selects the best k successors from the complete list and repeats the same process.

- In random-restart search where each search process runs independently, but in local beam search, the necessary information is shared between the parallel search processes.

Q.26. Enlist disadvantages of Local Beam Search.

Ans. Disadvantages of Local Beam Search:

(i) This search can suffer from a lack of diversity among the k states.

(ii) It is an expensive version of Hill Climbing Search.

Note: A variant of Local Beam Search is Stochastic Beam Search which selects k successors at random rather than choosing the best k successors.

BACKTRACKING

Q.27. What is Backtracking? Enlist its types Backtracking.

Ans. Backtracking:

Backtracking can be defined as General Algorithmic technique that considers searching every possible combination in order to solve a computational problem.

There are three types of problems in Backtracking:

(i) **Decision Problem:**

In Decision Problem, we search for a feasible solution.

(ii) **Optimization Problem:**

In Optimization Problem, we search for the best solution.

(iii) **Enumeration Problem:**

In Enumeration Problem, we find all feasible solutions.

Q.28. How to determine if a problem can be solved using Backtracking?

Ans. Determination of problem solved by Backtracking:

- Generally, every constraint satisfaction problem which has clear and well-defined constraints on any objective solution, that incrementally builds candidate to the solution and abandons a candidate "Backtracks" as soon as it determines that the candidate cannot possibly be completed to a valid solution, can be solved by Backtracking.
- However, most of the problems that are discussed, can be solved using other known Algorithms like Dynamic Programming or Greedy Algorithms in logarithmic, linear, linear-logarithmic time complexity in order of input size and therefore, outshine the backtracking algorithm in every respect (since backtracking algorithms are generally exponential in both time and space). However, a few problems still remain, that only have Backtracking Algorithms to solve them until now.
- Consider a situation that you have three boxes in front of you and only one of them has a gold coin in it but you do not know which one. So, in order to get the coin, you will have to open all of the boxes one by one.
- You will first check the first box, if it does not contain the coin, you will have to close it and check the second box and so on until you find the coin. This is what backtracking is, that is solving all sub-problems one by one in order to reach the best possible solution.

Working of Backtracking Algorithm:

The Algorithm begins to build up a solution, starting with an empty solution set. S = {}

(1) Add to Backtracking | Set 1Backtracking | Set 1the first move that is still left (All possible moves are added to one by one). This now creates a new sub-tree in the search tree of the algorithm.

(2) Check if satisfies each of the constraints in.

- If Yes, then the sub-tree is "Eligible" to add more "children".
- Else, the entire sub-tree is useless, so recurs back to step 1 using argument.

(3) In the event of "Eligibility" of the newly formed sub-tree, recurs back to step 1, using argument.

(4) If the check for returns that it is a solution for the entire data. Output and terminate the program.

If not, then return that no solution is possible with the current and hence discard it.

Q.29. Differentiate between Recursion and Backtracking.

Ans. Difference between Recursion and Backtracking:

In Recursion, the function calls itself until it reaches a base case. In Backtracking, we use recursion to explore all the possibilities until we get the best result for the problem.

Pseudo Code for Backtracking:

(1) **Recursive Backtracking Solution.**

```
void findSolutions(n, other params):
if (found a solution):
solutionsFound = solutionsFound + 1;
displaySolution();
if (solutionsFound >= solutionTarget):
System.exit(0);
return
for (val = first to last):
```

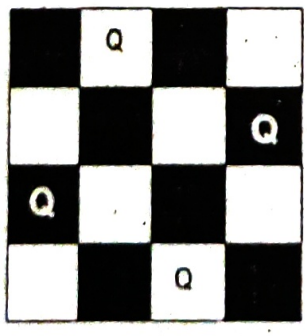
```
if (isValid(val, n)):
applyValue(val, n);
findSolutions(n+1, other params);
removeValue(val, n);
```

(2) Finding whether a solution exists or not

```
boolean findSolutions(n, other params):
if (found a solution):
displaySolution();
return true;
for (val = first to last):
if (isValid(val, n)):
applyValue(val, n);
if (findSolutions(n+1, other params))
return true;
removeValue(val, n);
return false;
```

Let us try to solve a standard Backtracking problem, N-Queen Problem.

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.



The expected output is a binary matrix which has 1s for the blocks where queens are placed. For example, following is the output matrix for the above 4 queen solution.

{ θ, 1, θ, θ}
{ θ, θ, θ, 1}
{ 1, θ, θ, θ}
{ θ, θ, 1, θ}

Q.30. Explain the Backtracking Algorithm.

Ans. Backtracking Algorithm:

- The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens.
- In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

Algorithms of Backtracking:

(1) Generate k-ary Strings
(2) Graph Coloring Problem
(3) Hamiltonian Cycles
(4) N-Queens Problem
(5) Knapsack Problem
(6) Start in the leftmost column
(7) If all queens are placed return true
(8) Try all rows in the current column. Do following for every tried row.
(a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
(b) If placing the queen in [row, column] leads to a solution then return true.
(c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
(4) If all rows have been tried and nothing worked, return false to trigger backtracking.

SEARCHING WITH PARTIAL OBSERVATION

Q.31. Explain Searching in partially observable environments.

Ans. Searching in Partially Observable Environments:

- Many problems cannot be solved without sensing. For example, the sensorless 8-puzzle is impossible. On the other hand, a little bit of sensing can go a long way: we can solve 8-puzzles if we can see just the upper-left corner square. The solution involves moving each tile in turn into the observable square and keeping track of its location from then on.
- For a partially observable problem, the problem specification will specify a PERCEPT(s) function that returns the percept received by the agent in a given state. If sensing is non-deterministic, then we can se PERCEPTS function that returns a set of possible percepts. For fully observable problems, PERCEPT(s) = s for every state s and for sensorless problems PERCEPT (s) = null.
- Consider a local-sensing vacuum world, in which the agent has a position sensor that yields the percept L in the left square and R in the right square and a dirt sensor that yields Dirty when the current square is dirty and Clean when it is clean. Thus, the PERCEPT in state 1 is [L, Dirty].
- With partial observability, it will usually be the case that several states produce the same percept; state 3 will also produce [L, Dirty]. Hence, given this initial percept, the
- initial belief state will be {1, 3}. We can think of the transition model between belief states for partially observable problems as occurring in three stages, as shown in Figure.

(i) Stage-1:

The prediction stage computes the belief state resulting from the action, RESULT(b, a), exactly as we did with sensorless problems. To emphasize that this is prediction, we use the

notation $\hat{b}$ = RESULT (b, a), where the "hat" over the b means "estimated" and we also use PERFECT (b, a) as a synonym for RESULT (b, a).

(ii) Stage-2:

The possible percepts stage computes the set of percept that could be observed in the predicted belief state (using the letter a for observation):

$$\text{POSSIBLE - PERCEPTS}\,\hat{b} = \{o : o = \text{PERFECT}(s) \text{ AND } s \in \hat{b}\}$$

(iii) Stage-3:

The update stage computes, for each possible percept, the belief state that would result from the percept. The updated belief state b_a is the set of states $\hat{b}$ that could have produced the percept:

$$b_a = \text{UPDATE}(\hat{b}, a) = (s : o = \text{PERCEPT}(s) \text{ and } s \in \hat{b}\}$$

- The agent needs to deal with possible percepts at planning time, because it won't know the actual percepts until it executes the plan. Notice that nondeterminism in the physical environment can enlarge the belief state in the prediction stage, but each updated belief state b_a can be no larger than the predicted belief state $\hat{b}$, observations can only help reduce uncertainty.
- Moreover, for deterministic sensing, the belief states for the different possible percepts will be disjoint, forming a partition of the original predicted belief state.
- Putting these three stages together, we obtain the possible belief states resulting from a given action and the subsequent possible percepts:

$$\text{RESULT}(b,a) = \{b_o : b_o = \text{UPDATED}(\text{PREDICT}(b, a), o) \text{ and } o \in \text{POSSIBLE-PERCEPTS}(\text{PREDICT}(b, a))\}$$

PERFORMANCE OF SEARCH ALGORITHMS

Q.32. Explain the performance of Search Algorithm.

Ans. Performance of Search Algorithm:

Algorithm	Time	Memory	Complete	Optimal
Breadth First	$O(b^d)$	$O(b^d)$	Yes	Yes
Depth First	$O(b^d)$	O(d)	No	No
DF iterative deepening	O(bd)	O(d)	Yes	Yes
Bidirectional	$O(b^{d/2})$	$O(b^{d/2})$	Yes	Yes
Hill Climbing	$O(b^d)$	$O(1)$- $O(b^d)$	No	No
Best First	$O(b^d)$	$O(b^d)$	Yes	No
A*	$O(b^d)$	$O(b^d)$	Yes	Yes
Beam Search	$O(n^d)$	$O(n^d)$	No	No
Means End	$O(b^d)$	$O(b^d)$	No	No

REASONING SYSTEMS

Q.33.What is Reasoning? Explain the various method of Reasoning.

Ans. Reasoning:

- Reasoning plays a great role in the process of Artificial Intelligence. Thus Reasoning can be defined as the logical process of drawing conclusions, making predictions or constructing approaches towards a particular thought with the help of existing knowledge.
- In Artificial Intelligence, reasoning is very important because to understand the human brain, how the brain thinks, how it draws conclusions towards particular things for all these sorts of works we need the help of reasoning.

Methods of Reasoning:

The reasoning is classified into the following types:

(1) Deductive Reasoning:

- Deductive Reasoning is the strategic approach that uses available facts, information or knowledge to draw valid conclusions. It basically beliefs in the facts and ideas before drawing any result.
- Deductive Reasoning uses a top-down approach. In Deductive Reasoning, the arguments can be valid or invalid based on the value of the premises.
- If the value of the premises is true, then the conclusion is also true. Deductive Reasoning helps in scanning the generalized statement into a valid conclusion.

Some of the examples of Deductive Reasoning are as follows:

(i) People who are aged 20 or above are active users of the internet.

(ii) Out of the total number of students present in the class, the ratio of boys is more than the girls.

(2) Inductive Reasoning:

- Inductive Reasoning is completely different from the Deductive Reasoning approach because Inductive Reasoning is associated with the hypothesis generating approach rather than drawing any particular conclusion to the facts at the beginning of the process.
- Inductive reasoning help in making generalization from specific facts and knowledge. Inductive Reasoning is the bottom-up process.
- In Inductive Reasoning even if the premises are true there is no chance that the conclusion will also be true because it depends upon the inductive argument which can be either strong or weak.

Some of the examples of Inductive Reasoning are as follows:

(i) All the students present in the classroom are from London.

(ii) Always the hottest temperature is recorded in Death Valley.

(3) Common Sense Reasoning:

- Common Sense Reasoning is the most occurred type of Reasoning in daily life events. It is the type of reasoning which comes from experiences. When a human face a different situation in life it gain some knowledge.
- So, whenever in the next point of time it faces a similar type of situation then it uses its previous experiences to draw a conclusion to do situation.

Some of the examples are:

(i) When a bike crosses the traffic signal when it is red then it learns from its mistakes and next time the bike is aware of the signal and actions.

(ii) While overtaking someone on the road what all ideas should be kept in mind.

(4) Monotonic Reasoning:

- Monotonic Reasoning is the type of Reasoning which follows a different approach towards the thinking process it uses facts, information and knowledge to draw a conclusion about the problem but the major point is its conclusion remain fixed permanently once it is decided because even if we add new information or facts to the existing one the conclusion remains the same it doesn't change.
- Monotonic Reasoning is used mainly in conventional reasoning systems and logic-based systems.

Some Examples of Monotonic Reasoning are as follows:

(i) The Sahara desert of the world is one of the most spectacular deserts.

(ii) One of the longest rivers in the world is the Nile River.

(5) Abductive Reasoning:

- Abductive Reasoning is a type of reasoning which acts differently from all the above reasoning strategies. It begins with an incomplete set of facts, information and knowledge and then proceeds to find the most deserving explanation and conclusion.
- It draws conclusions based on what facts you know at present rather than collecting some outdated facts and information. It mostly plays a great role in the daily life decision making process.

Some of the examples of Abductive Reasoning are as follows:

(i) Doctor drawing conclusions regarding your health based on test reports.

(ii) A bowl of soup is kept and vapour evaporating from it which draws the conclusion that the bowl is hot in nature.

PLANNING

Q.34. What is a Plan? Give types of Planning.

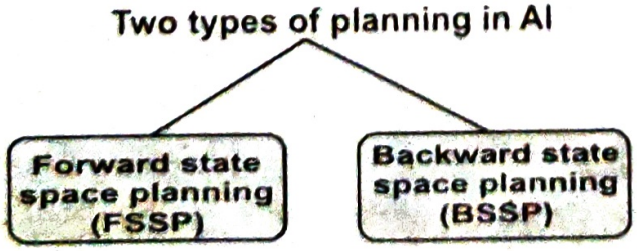
Ans. Plan:

- We require domain description, task specification, and goal description for any planning system.
- A plan is considered a sequence of actions, and each action has its preconditions that must be satisfied before it can act and some effects that can be positive or negative.

There is two types of Planning at basic level:

(i) Forward State Space Planning (FSSP).

(ii) Backward State Space Planning (BSSP).



(i) Forward State Space Planning (FSSP):

FSSP behaves in the same way as forwarding state-space search. It says that given an initial state S in any domain, we perform some necessary actions and obtain a new state S' (which also contains some new terms), called a progression. It continues until we reach the target position. Action should be taken in this matter.

Advantage:

The algorithm is Sound.

Disadvantage:

Large branching factor.

(ii) Backward State Space Planning (BSSP):

BSSP behaves similarly to backward state-space search. In this, we move from the target state g to the sub-goal g, tracing the previous action to achieve that goal. This process is called regression (going back to the previous goal or sub-goal). These sub-goals should also be checked for consistency. The action should be relevant in this case.

Advantage:

Small branching factor (much smaller than FSSP).

Disadvantages:

Not sound algorithm (sometimes inconsistency can be found).

So for an efficient planning system, we need to combine the features of FSSP and BSSP, which gives rise to target stack planning which will be discussed in the next article.

Q.35. What is Planning in AI?

Ans. Planning:

- Planning in Artificial Intelligence is about decision-making actions performed by robots or computer programs to achieve a specific goal.
- Execution of the plan is about choosing a sequence of tasks with a high probability of accomplishing a specific task.

Q.36. What is the Role of Planning in Artificial Intelligence?

Ans. Role of planning in Artificial Intelligence:

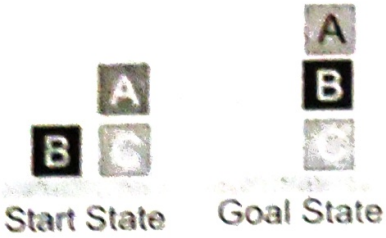
- Artificial Intelligence is an important technology in the future. Whether it is intelligent robots, self-driving cars, or smart cities, they will all use different aspects of Artificial Intelligence. But Planning is very important to make any such AI project.
- Even Planning is an important part of Artificial Intelligence which deals with the tasks and domains of a particular problem. Planning is considered the logical side of acting.
- Everything we humans do is with a definite goal in mind and all our actions are oriented towards achieving our goal. Similarly, Planning is also done for Artificial Intelligence.
- For Example Planning is required to reach a particular destination. It is necessary to find the best route in Planning, but the tasks to be done at a particular time and why they are done are also very important.
- That is why Planning is considered the logical side of acting. In other words, Planning is about deciding the tasks to be performed by the Artificial Intelligence System and the system's functioning under domain-independent conditions.

Q.37. What is Block-world Planning Problem?

Ans. Block-world Planning Problem:

- The block-world problem is known as the Sussmann anomaly.
- The non-interlaced planners of the early 1970s were unable to solve this problem. Therefore it is considered odd.
- When two sub-goals, G1 and G2, are given, a non-interleaved planner either produces a plan for G1 that is combined with a plan for G2 or vice versa.
- In the block-world problem, three blocks labeled 'A', 'B', and 'C' are allowed to rest on a flat surface. The given condition is that only one block can be moved at a time to achieve the target.

- The start position and target position are shown in the following diagram:



World planning problem

Q.38. What are the important steps to make plan?

OR Enlist the components of Planning System.

Ans. Components of the Planning System:

The plan includes the following important steps:

(i) Choose the best rule to apply the next rule based on the best available guess.

(ii) Apply the chosen rule to calculate the new problem condition.

(iii) Find out when a solution has been found.

(iv) Detect dead ends so they can be discarded and direct system effort in more useful directions.

(v) Find out when a near-perfect solution is found.

Q.39. Describe Target Stack Plan in detail.

Ans. Target Stack Plan:

- Target Stack Plan is one of the most important planning algorithms used by STRIPS.
- Stacks are used in algorithms to capture the action and complete the target. A knowledge base is used to hold the current situation and actions.
- A target stack is similar to a node in a search tree, where branches are created with a choice of action.

The important steps of the algorithm are mentioned below:

(i) Start by pushing the original target onto the stack. Repeat this until the pile is empty. If the stack top is a mixed target, push its unsatisfied sub-targets onto the stack.

(ii) If the stack top is a single unsatisfied target, replace it with action and push the action precondition to the stack to satisfy the condition.

(iii) If the stack top is an action, pop it off the stack, execute it and replace the knowledge base with the action's effect.

(iv) If the stack top is a satisfactory target, pop it off the stack.

Q.40. Explain Non-linear Planning in detail.

Ans. Non-linear Planning:

Non-linear Planning is used to set a goal stack and is included in the search space of all possible sub-goal orderings. It handles the goal interactions by the interleaving method.

Advantages of Non-linear Planning:

Non-linear Planning may be an optimal solution concerning planning length (depending on the search strategy used).

Disadvantages of Non-linear Planning:

It takes a larger search space since all possible goal orderings are considered.

Complex Algorithm to understand Algorithm:

(i) Choose a goal 'g' from the goal set

(ii) If 'g' does not match the state, then

- Choose an operator 'o' whose add-list matches goal g
- Push 'o' on the OpStack
- Add the preconditions of 'o' to the goal set

(iii) While all preconditions of the operator on top of OpenStack are met in a state

- Pop operator o from top of opstack
- state = apply(o, state)
- plan = [plan; o]

PLANNING WITH STATE-SPACE SEARCH

Q.41. Explain Planning with State Space Search.

Ans. Planning with State-Space Search:

The most straight forward approach is to use state-space search. Because the descriptions of actions in a planning problem specify both preconditions and effects, it is possible to search in either direction i.e. forward from the initial state or backward from the goal, as shown in Figure given below. We can also use the explicit action and goal representations to derive effective heuristics automatically.

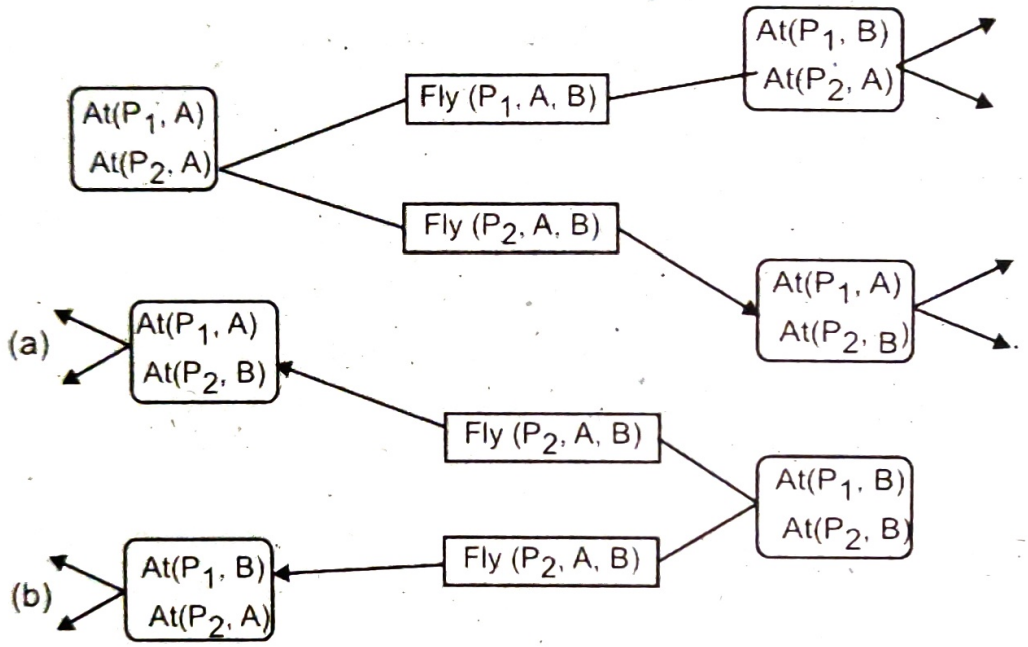


Fig. Two approaches to searching for a plan.

(a) Forward (Progression) state-space search, starting in the initial state and using the problems action to search forward for the goal state, (b) Backward (regression) state-space search a belief -state search starting at the goal state(s) and using the inverse of the actions to search backward for the initial state.

(1) Forward State-Space Search:

- Planning with forward state-space search is similar to the problem-solving approach. It is sometimes called progression planning, because it moves in the forward direction.
- We start with the problem's initial state, considering sequences of actions until we reach a goal state.

- The formulation of planning problem as state-space search problems is as follows:

(i) The initial state of the search is the initial state from the planning problem. In general each state will be set of positive ground literals; literals not appearing are false.

(ii) The actions which are applicable to a state are all those whose preconditions are satisfied. The successor state resulting from an action is generated by adding the positive effect literals and deleting the negative effect literals.

(iii) The goal test checks whether the state satisfies the goal of the planning problem.

(iv) The step cost of each action is typically 1. Although it would be easy to allow different costs for different actions, this was seldom done by STRIPS planners.

- Since function symbols are not present, the state space of a planning problem is finite and therefore, any graph search algorithm such as A * will be a complete planning algorithm.
- From the early days of planning research it is known that forward state-space search is too inefficient to be practical. Mainly, this is because of a big branching factor since forward search does not address only relevant actions, (all applicable actions are considered).

For Example:

An air cargo problem with 10 airports, where each airport has 5 planes and 20 pieces of cargo.

- The goal is to move all the cargo at airport A to airport B. There is a simple solution to the problem: load the 20 pieces of cargo into one of the planes at A, fly the plane to B and unload the cargo.
- But finding the solution can be difficult because the average branching factor is huge: each of the 50 planes can fly to 9 other airports and each of the 200 packages can be either

unloaded (if it is loaded), or loaded into any plane at its airport (if it is unloaded).

- On average, let's say there are about 1000 possible actions, so the search tree up to the depth of the obvious solution has about 1000 nodes. It is thus clear that a very accurate heuristic will be needed to make this kind of search efficient.

(2) Backward State-Space Search:

- Backward search can be difficult to implement when the goal states are described by a set of constraints which are not listed explicitly. In particular, it is not always obvious how to generate a description of the possible predecessors of the set of goal states.
- The STRIPS representation makes this quite easy because sets of states can be described by the literals which must be true in those states.
- The main advantage of backward search is that it allows us to consider only relevant actions. An action is relevant to a conjunctive goal if it achieves one of the conjuncts of the goal.

For Example:

The goal in our 10-airport air cargo problem is to have 20 pieces of cargo at airport B, or more precisely.

$A_t (C_1 B) \wedge A_t (C_2 B)$.....$A_t (C_{20} B)$

Now consider the conjunct $A_t (C_1 B)$ Working backwards, we can seek those actions which have this as an effect,

There is only one:

Unload (C_1, p, B), where plane p is unspecified.

- We may note that there are many irrelevant actions which can also lead to a goal state. For example, we can fly an empty plane from Mumbai to Chennai; this action reaches a goal state from a predecessor state in which the plane is at Mumbai and all the goal conjuncts are satisfied. A backward

search which allows irrelevant actions will still be complete, but it will be much less efficient. If a solution exists, it should be found by a backward search which allows only relevant action.

- This restriction to relevant actions only means that backward search often has a much lower branching factor than forward search. For example, our air cargo problem has about 1000 actions leading forward from the initial state, but only 20 actions working backward from the goal. Hence backward search is more efficient than forward searching.
- Searching backwards is also called Regression Planning. The principal question in regression planning is: what are the states from which applying a given action leads to the goal? Computing the description of these states is called regressing the goal through the action. To see how does it work, once again consider the air cargo example. We have the goal

  $A_t (C_1 B) \wedge A_t (C_2 B) \wedge ..... \wedge A_t (C_{20} B)$
- The relevant action UNLOAD$^{\cdot}$ (C_1, p, B), achieves the first conjunct. The action will work only if its preconditions are satisfied. Therefore, any predecessor state must include these preconditions: In $(C_1, p) \wedge A_t (p B)$ as sub-goals. Moreover, the sub-goal At (C_1, B) should not be true in the predecessor state which will no doubt be a goal but not relevant one (justify).

  Thus, the Predecessor Description is:

  In $C_1, p) \wedge A_t(p; B) \wedge A_t(C_2, B) \wedge ..... \wedge A_t (C_{20}, B)$
- In addition to insisting that actions achieve some desired literal, we must insist that the actions do not undo any desired literals. An action which satisfies this restriction is called consistent. For example, the action load (C_2, p) would not be consistent with the current goal, because it would negate the literal $A_t (C_2, p)$ (verify).

- Given definitions of relevance and consistency, we can now describe the general process of constructing predecessors for backward search. Given a goal description G, let A be an action which is relevant and consistent.

The corresponding predecessor is constructed as follows:

(I) Any positive effects of A which appear in G are deleted.

(II) Each precondition literal of A is added, unless it already appears.

Any of the standard search algorithms can be used to carry out the search. Termination occurs when a predecessor description is generated which is satisfied by the initial state of the planning problem. In the first-order logic, satisfaction might require a substitution for variables in the predecessor description. For example, the predecessor description in the preceding paragraph is satisfied by the initial state.

In $(C_1, P_{12}) \wedge A_t (P_{12}, B) \wedge A_t (C_2, B) \wedge \ldots\ldots \wedge A_t (C_{20}, B)$ with substitution (P / P_{12}) The substitution must be applied to the action leading from the state to the goal, producing the solution

[Unload C_1,P_{12}, B)]

Q.42. Explain Heuristics for State-Space Search in detail.

Ans. Heuristics for State-Space Search:

- It turns out that neither forward nor backward search is efficient without a good heuristic function. Let us recall that in a search technique a heuristic function estimates the distance from a state to the goal. In STRIPS planning, the cost of each action is 1, so the distance is the number of actions.
- The basic idea is to look at the effects of the actions and to guess how many actions are needed to achieve all the goals. Finding the exact number is NP hard, but it is possible to find

reasonable estimates, most of the time without too much computation. We might also be able to derive an admissible heuristic- (one which does not overestimate). This could be achieved with A * search.

- There are two approaches which can be tried. The first is to derive relaxed problem from the given problem specification. The optimal solution cost for the relaxed problem - which is very easy to solve-gives an admissible heuristic for the admissible original problem. A problem with fewer restrictions on the actions is called a relaxed problem.
- The cost of an optimal solution to a relax, problem is an admissible heuristic for the original problem. The heuristic is admissible because the optimal solution in the original problem, is by definition, also a solution in the relaxed problem and therefore must be at least as expensive as the optimal solution in the relaxed problem. The second approach is to pretend that a pure divide-and-conquer algorithm will work.
- This is called the sub-goal independence assumption: the cost of solving a conjunction of sub-goals in approximated by the sum of the costs of solving each sub-goal independently. The sub-goal independence assumption can be optimistic or pessimistic.
- It is optimistic when there are negative interactions between the sub-plans for each sub-goal-for example, when an action in one sub-plan deletes a goal achieved by another sub-plan. It is pessimistic, and therefore inadmissible, when sub-plans contain redundant actions-for instance, two actions that could be replaced by a single action in the merged plan.
- So let us consider how to derive relaxed planning problems. Since explicit representations of preconditions and effects are available (compared with ordinary search space where the

successor states are not known) the process will work by modifying those representations.

- The simplest idea is to relax the problem by removing all preconditions from the actions. Then every action will always be applicable, and any literal can be achieved in one step, if there is an applicable action (goal is impossible if action is not applicable).
- It implies that the number of steps required to solve a conjunction of goals is the number of unsatisfied goals- almost but not quite, because:

(1) There may be two actions each which of deletes the goal literal achieved by the other, and

(2) Some actions may achieve multiple goals.

- If we combine our relaxed problem with the sub-goal independence assumption, both of these issues are assumed and the resulting heuristic is exactly the number of unsatisfied goals.
- In many cases, a more accurate heuristic is obtained by considering at least the positive interactions arising from actions which achieve multiple goals. First, we relax the problem further by removing negative effects. Then, we count the minimum number of actions required such that the union of those actions' positive effects satisfies the goal.

Q.43. Give a simple example of Heuristics for State-Space Search.

Ans. Example of Heuristics for State-Space Search:

Consider

Goal (A ∧ B ∧ C)

Action (X Effect: A A P)

Action (Y, Effect: B A C A Q)

Action (Z, Effect: B A P A Q)

- The minimal set cover of goal {A, B, C} is given by actions {X, Y}, so the set cover heuristic returns a cost of 2. This improves on the sub-goal independence assumption, which gives a heuristic value of 3. There is one minor irritation: the set cover problem is NP hard. A simple greedy-set cover algorithm is guaranteed to return a value which is within a factor of log n of the true minimum value, where n is the number of literals in the goal, and usually works much better than this. Unfortunately, the greedy algorithm loses the guarantee of admissibility for the heuristic.
- It is also possible to generate relaxed problem by removing negative effects without removing preconditions. That is, if an action has the effect $A \wedge \neg B$ in the original problem, it will have the effect A in the relaxed problem. This means that we need not worry about negative interactions between sub-plans, because no action can delete the literals achieved by another action.
- The solution cost of the resulting relaxed problem gives what is called the empty-delete-list heuristic. This heuristic is quite accurate, but computing it involves actually running a (simple) planning algorithm. In practice, the search in the relaxed problem is often fast enough that the cost is worthwhile.
- The heuristics described here can be used in either the progression or the regression direction. At present, progression planners using the empty-delete-list heuristic hold the lead. But it is likely to change as new heuristics and new search techniques are being explored. Since planning is exponentially hard, no algorithm will be efficient for all problems, but mostly practical problems can be solved with the heuristic method.

PARTIAL ORDER PLANNING

Q.44. Explain Partial-order Planning in detail with Algorithm.

Ans. Partial-order Planning:

- The forward and regression planners enforce a total ordering on actions at all stages of the planning process. The CSP planner commits to the particular time that the action will be carried out. This means that those planners have to commit to an ordering of actions that cannot occur concurrently when adding them to a partial plan, even if there is no particular reason to put one action before another.
- The idea of a partial-order planner is to have a partial ordering between actions and only commit to an ordering between actions when forced. This is sometimes also called a Non-linear Planner, which is a misnomer because such planners often produce a linear plan.
- A partial ordering is a less-than relation that is transitive and asymmetric. A partial-order plan is a set of actions together with a partial ordering, representing a "before" relation on actions, such that any total ordering of the actions, consistent with the partial ordering, will solve the goal from the initial state. Write $act_0 < act_1$ if action act_0 is before action act_1 in the partial order. This means that action act_0 must occur before action act_1.
- For uniformity, treat start as an action that achieves the relations that are true in the initial state, and treat finish as an action whose precondition is the goal to be solved. The pseudoaction start is before every other action, and finish is after every other action. The use of these as actions means that the algorithm does not require special cases for the initial situation and for the goals. When the preconditions of finish hold, the goal is solved.

- An action, other than start or finish, will be in a partial-order plan to achieve a precondition of an action in the plan. Each precondition of an action in the plan is either true in the initial state, and so achieved by start, or there will be an action in the plan that achieves it.
- We must ensure that the actions achieve the conditions they were assigned to achieve. Each precondition P of an action act1 in a plan will have an action act_0 associated with it such that act_0 achieves precondition P for act_1. The triple (act_0, P, act_1) is a causal link. The partial order specifies that action act0 occurs before action act_1, which is written as $act_0 < act_1$. Any other action A that makes P false must either be before act_0 or after act_1.
- Informally, a partial-order planner works as follows:

(i) Begin with the actions start and finish and the partial order start < finish.

(ii) The planner maintains an agenda that is a set of (P,A) pairs, where A is an action in the plan and P is an atom that is a precondition of A that must be achieved.

(iii) Initially the agenda contains pairs (G,finish), where G is an atom that must be true in the goal state.

- At each stage in the planning process, a pair (G, act_1) is selected from the agenda, where P is a precondition for action act_1. Then an action, act_0, is chosen to achieve P.
- That action is either already in the plan - it could be the start action, for example - or it is a new action that is added to the plan. Action act_0 must happen before act_1 in the partial order.
- It adds a causal link that records that act_0 achieves P for action act_1. Any action in the plan that deletes P must

happen either before act_0 or after act_1. If act_0 is a new action, its preconditions are added to the agenda and the process continues until the agenda is empty.

- This is a non-deterministic procedure. The "choose" and the "either ...or ..." form choices that must be searched over. There are two choices that require search.

Q.45. Which action is selected to achieved G.

OR Whether an action that deletes G happens before act_0 or after act_1.

Ans. Non-deterministic Procedure: Partial Order Planner (Gs)

1:
2: Inputs
3: Gs: set of atomic propositions to achieve
4: Output
5: linear plan to achieve Gs
6: Local
7: Agenda: set of (P,A) pairs where P is atom and A an action
8: Actions: set of actions in the current plan
9: Constraints: set of temporal constraints on actions
10: CausalLinks: set of (act_0, P, act_1) triples
11: Agenda $\leftarrow$ {(G,finish):G $\in$ Gs}
12: Actions $\leftarrow$ {start,finish}
13: Constraints $\leftarrow$ {start<finish}
14: CausalLinks $\leftarrow$ {}
15: repeat
16: select and remove (G, act_1) from Agenda
17: either
18: choose $act_0 \in$ Actions such that act_0 achieves G
19: or
20: choose act_0 (Actions such that act_0 achieves G

21: Actions ←Actions ∪{ act_0 }

22: Constraints ←add_const(start< act_0 ,Constraints)

23: for each CL ∈ CausalLinks do

24: Constraints ←protect(CL, act_0 ,Constraints)

25:

26: Agenda ←Agenda ∪{(P, act_0): P is a precondition of act_0 }

27:

28: Constraints←add_const (act_0 < act_1 1,Constraints)

29: CausalLinks ∪ { (act_0, G, act_1) }

30: for each A ∈ Actions do

31: Constraints ←protect (act_0, G, act_1) A,Constraints)

32:

33: until Agenda={}

34: return total ordering of Actions consistent with Constraints

The algorithm of Partial Order Planner is given above.

- The function add_const(act_0 < act_1 ,Constraints) returns the constraints formed by adding the constraint act_0 < act_1 to Constraints and it fails if act_0 < act_1 is incompatible with Constraints. There are many ways this function can be implemented.
- The function protect ((act_0, G, act_1),A, Constraints) checks whether A ≠ act_0 and A ≠ act_1 and A deletes G. If so, it returns either {A< act_0 } ∪ Constraints or { act_1 <A} ∪ Constraints. This is a non-deterministic choice that is searched over. Otherwise it returns Constraints.

Example:

Consider the goal ¬swc ∧ ¬mw, where the initial state contains RLoc=lab, swc, ¬rhc, mw, ¬rhm.

Initially the agenda is

(¬swc,finish), (¬mw,finish).

Suppose (¬swc,finish) is selected and removed from the agenda. One action exists that can achieve ¬swc, namely deliver coffee, dc, with preconditions off and rhc. At the end of the repeat loop, Agenda contains
(off,dc),(rhc,dc),(¬mw,finish).
Constraints is {start<finish, start < dc, dc <finish}. There is one causal link, (dc,¬swc,finish). This causal link means that no action that undoes ¬swc is allowed to happen after dc and before finish.
Suppose (¬mw,finish) is selected from the agenda. One action exists that can achieve this, pum, with preconditions mw and RLoc=mr. The causal link (pum, ¬mw,finish) is added to the set of causal links; (mw,pum) and (mr,pum) are added to the agenda.
Suppose (mw,pum) is selected from the agenda. The action start achieves mw, because mw is true initially. The causal link (start,mw,pum) is added to the set of causal links. Nothing is added to the agenda.
At this stage, there is no ordering imposed between dc and pum.
Suppose (off,dc) is removed from the agenda. There are two actions that can achieve off: mc_cs with preconditions cs and mcc_lab with preconditions lab. The algorithm searches over these choices. Suppose it chooses mc_cs. Then the causal link (mc_cs,off,dc) is added.
The first violation of a causal link occurs when a move action is used to achieve (mr,pum). This action violates the causal link (mc_cs,off,dc) and so must happen after dc (the robot goes to the mail room after delivering coffee) or before mc_cs.

- The preceding algorithm has glossed over one important detail. It is sometimes necessary to perform some action more

than once in a plan. The preceding algorithm will not work in this case, because it will try to find a partial ordering with both instances of the action occurring at the same time.

- To fix this problem, the ordering should be between action instances and not actions themselves.
- To implement this, assign an index to each instance of an action in the plan and the ordering is on the action instance indexes and not the actions themselves. This is left as an exercise.

PLANNING AND ACTING IN THE REAL WORLD

Q.46. What is role of Time, Schedule and Resources in Planning and Acting in Real World?

Ans. Time, Schedules and Resources in Planning:

- Time is of the essence in the general family of applications called Job Shop Scheduling. Such tasks require completing a set of jobs, each of which consists of a sequence of actions, where each action has a given duration and might require some resources.
- The problems is to determine a schedule that minimizes the total time required to complete all the jobs, while respecting the resource constraints.

Example of Job Shop Scheduling Problem:

- This is a highly simplified automobile assembly problem. There are two jobs i.e. assembling cars C_1 and C_2. Each job consists of three actions which are as follows:

(i) Adding the engine.

(ii) Adding the wheels.

(iii) Inspecting the results.