1.3 PROBLEM SOLVING AGENTS

1.3.1 Intelligent Agents

• An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

• For example consider human as agent. Human has eyes, ears and other organs which are sensors. Hands, legs, mouth and other body part work as actuators.

• Lets consider another example of agent-Robot. A Robotic agent might have cameras, infrared rangefinders as sensors. Robot can have various motors for actuators.

More examples of agent

1. Agent: Software agent

Sensors: Keystrokes, file contents and network packets.

Actuator: Screen, writing files, network packet.

2. Agent: Internet shopping agent

Sensors: HTML, , pages (text graphics script)

Actuators: Forms, display to user, follow URL.

1.3.2 The Al Terminology

1) Percept

The term percept refers to the agent's perceptual inputs at any given instant. Examples -

1) A human agent percepts "Bird flying in the sky "through eyes and takes i (photograph)".

2) A robotic agent perceive "Temperature of a boiler" through cameras and takes the control action.

2) Percept Sequence

An agent's percept sequence is the complete history of everything the agent has ever perceived. Agent has choice of action at any given instant and it can depend on the entire percept sequence agent has recorded.

3) Agent Function

It is defined as mathematical function which maps each and every possible percept sequence to a possible action.

This function has input as percept sequence and it gives output as action.

Agent function can be represented in a tabular form.

Example -

ATM machine is agent, it display menu for withdrawing money, when ATM card is inserted. When provided with percept sequence (1) A transaction type and (2) PIN number, then only user gets cash.

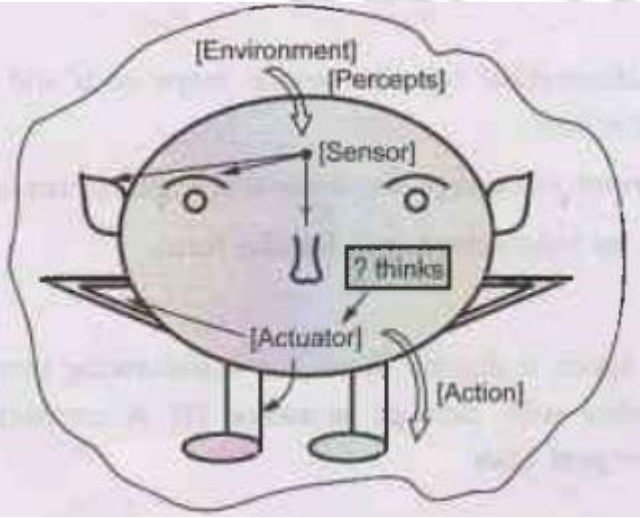
4) Agent Program

When we want to develop agent program we need to tabulate all the agent functions that describes any given agent.

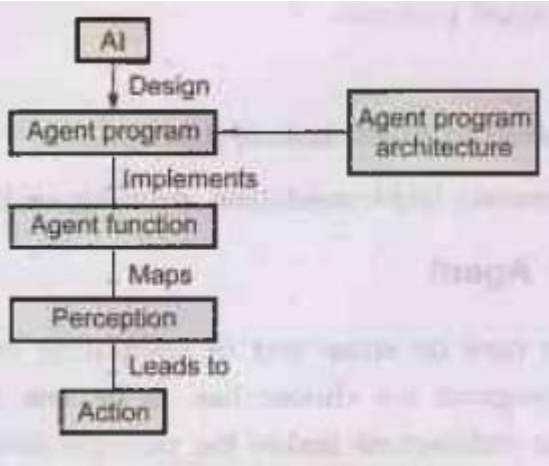
1.3.3 Architecture of Agent

• The agent program runs on some sort of computing device, which is called the architecture. The program we choose has to be one that the architecture will accept and run. The architecture makes the percepts from the sensors available to the program, runs the program and feeds the program's action choices to the effectors as they are generated. The relationship among agents, architectures and programs can be summed up as follows:

Agent = Architecture + Program



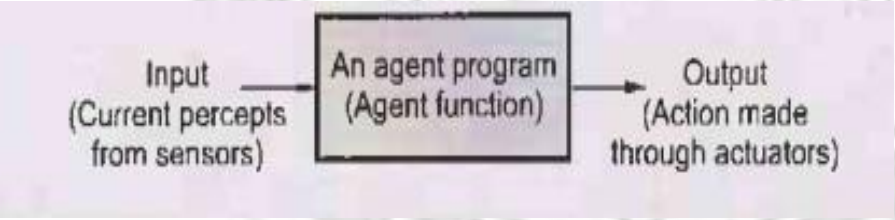
• Following diagram illustrates the agent's action process, as specified by architecture. This can be also termed as agent's structure.



1.3.4 Role of an Agent Program

• An agent program is internally implemented as agent function.

• An agent program takes input as the current percept from the sensor and return an action to the effectors (Actuators).



Weak and Strong Agent

• An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors/actuators.

Weak Agent

• A weak notion says that an agent is a hardware or software based computer system that has the following properties:

1] Autonomy

Agents operate without direct intervention of humans and have control over their actions and internal state.

2] Social ability

Agents interact with other agents (and possibly humans) via an agent communication language.

3] Reactivity

Agents perceive their environment and respond in timely and rational fashion to changes that occur in it.

4] Pro-activeness

Agents do not simply act in response to their environment, they are capable of taking the initiative, generate their own goals and act to achieve them.

S**trong Agent**

A stronger nation says that an agent has mental properties, such as knowledge, belief, intention, obligation. In addition, and agent has other properties such as: -

1. Mobility : Agents can move around from one machine to another and across different system architectures and platforms.

2. Veracity: Agents do not knowingly communicate false information.

3. Rationality: Agents will try to achieve their goals and not acts in such a way that would prevent their goals from being achieved.

1.3.5 PROBLEM-SOLVING APPROACH IN ARTIFICIAL INTELLIGENCE PROBLEMS

The **reflex agents** are known as the simplest agents because they directly map states into actions. Unfortunately, these agents fail to operate in an environment where the mapping is too large to store and learn. **Goal-based agent,** on the other hand, considers future actions and the desired outcomes.

Here, we will discuss one type of goal-based agent known as a **problem-solving agent**.

Problem-solving agent

The problem-solving agent perfoms precisely by defining problems and its several solutions.

- According to psychology, *"a problem-solving refers to a state where we wish to reach to a definite goal from a present state or condition."*
- According to computer science, *a problem-solving is a part of artificial intelligence which encompasses a number of techniques such as algorithms, heuristics to solve a problem.*

Therefore, a problem-solving agent is a **goal-driven agent** and focuses on satisfying the goal.

1.3.6 PROBLEM DEFINITION

To build a system to solve a particular problem, we need to do four things:

(i) **Define** the problem precisely. This definition must include specification of the initial situations and also final situations which constitute (i.e) acceptable solution to the problem.

(ii) **Analyze** the problem (i.e) important features have an immense (i.e) huge impact on the appropriateness of various techniques for solving the problems.

(iii) **Isolate and represent** the knowledge to solve the problem.

(iv) **Choose the best** problem - solving techniques and apply it to the particular problem.

Steps performed by Problem-solving agent

- **Goal Formulation:** It is the first and simplest step in problem-solving. It organizes the steps/sequence required to formulate one goal out of multiple goals as well as actions to achieve that goal. Goal formulation is based on the current situation and the agent's performance measure (discussed below).
- **Problem Formulation:** It is the most important step of problem-solving which decides what actions should be taken to achieve the formulated goal. There are following five components involved in problem formulation:
  - **Initial State:** It is the starting state or initial step of the agent towards its goal.
  - **Actions:** It is the description of the possible actions available to the agent.
  - **Transition Model:** It describes what each action does.
  - **Goal Test:** It determines if the given state is a goal state.
  - **Path cost:** It assigns a numeric cost to each path that follows the goal. The problem-solving agent selects a cost function, which reflects its performance measure. Remember, **an optimal solution has the lowest path cost among all the solutions.**

Note: Initial state, actions, and **transition model** together define the **state-space** of the problem implicitly. State-space of a problem is a set of all states which can be reached from the initial state followed by any sequence of actions. The state-space forms a directed map or graph where nodes are the states, links between the nodes are actions, and the path is a sequence of states connected by the sequence of actions.

- **Search:** It identifies all the best possible sequence of actions to reach the goal state from the current state. It takes a problem as an input and returns solution as its output.

- **Solution:** It finds the best algorithm out of various algorithms, which may be proven as the best optimal solution.
- **Execution:** It executes the best optimal solution from the searching algorithms to reach the goal state from the current state.

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.

1.3.7 Toy Problem:

Consider following example, A BLACK BALLS PICKER

Picker World (Environment)

It is a simple and made-up world so one can invent many variations.

It has two buckets at two locations, L1 and L2 (for simplicity consider square area for location), full of BLACK and WHITE colour balls.

Picker and its Perceptions

Picker perceives at which location it is. It can perceive that, is there a BLACK ball at the given location.

Agent Actions

Picker can choose to MOVE LEFT or MOVE RIGHT, PICK UP BLACK BALL or be ideal that as do nothing.

A function can be devised as follows - If the current location bucket has more BLACK BALLS then PICK, otherwise MOVE to other square.

Diagram Depicting Black Ball Picker

Following is the partial tabulation of a simple agent function for the black ball picker.

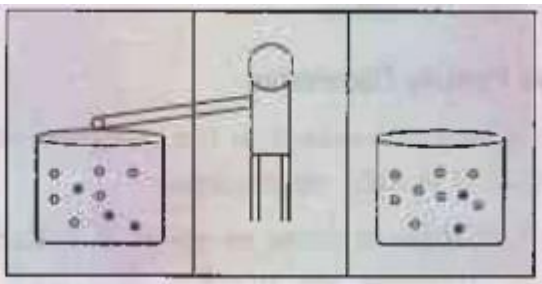
- 

Fig. 2.9.1 Black ball picker world with two buckets at two locations

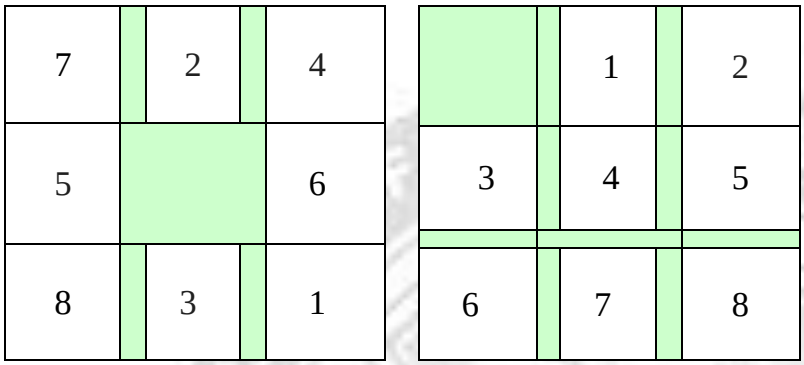
-

Percept Sequence	Action
[L1, No Black Ball]	Right
[L1, More Black Balls]	Pick
[L2, No Black Ball]	Right
[L2, More Black Balls]	Pick
:	
:	
[L1, No Black Ball], [L1, No Black Ball]	Right
[L1, No Black Ball], [L1, More Black Balls]	Pick
:	
:	
[L1 No Black Ball], [L1, No Black Ball],	
[L1, No Black Ball]	Right
[L1, No Black Ball], [L1, No Black Ball],	
[L1, More Black Balls]	Pick
:	
:	

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:

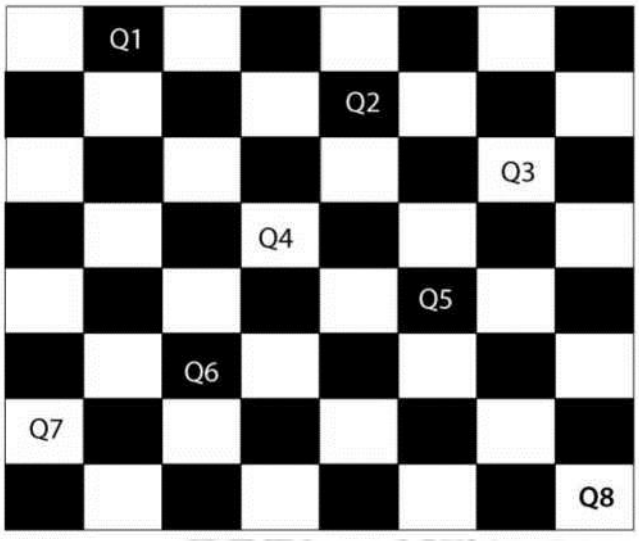
- **States:** It describes the location of each numbered tiles and the blank tile.
- **Initial State:** We can start from any state as the initial state.

- **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.

  Note: The 8-puzzle problem is a type of **sliding-block problem** which is used for testing new search algorithms in artificial intelligence.

- **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:

1. **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.

2. **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 x 10^{14}** to **2057**, and it is easy to find the solutions.

Some Real-world problems

- **Traveling 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.
- **VLSI Layout problem:** In this problem, millions of components and connections are positioned on a chip in order to minimize the area, circuit-delays, stray-capacitances, and maximizing the manufacturing yield.

The layout problem is split into two parts:

- **Cell layout:** Here, the primitive components of the circuit are grouped into cells, each performing its specific function. Each cell has a fixed shape and size. The task is to place the cells on the chip without overlapping each other.
- **Channel routing:** It finds a specific route for each wire through the gaps between the cells.

- **Protein Design:** The objective is to find a sequence of amino acids which will fold into 3D protein having a property to cure some disease.

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.**

1.4 SEARCH STRATEGIES

There are two types of strategies that describe a solution for a given problem:

1. ***Uninformed Search (Blind Search)***

This type of search strategy does not have any additional information about the states except the information provided in the problem definition. They can only generate the successors and distinguish a goal state from a non-goal state. These type of search does not maintain any internal state, that's why it is also known as **Blind search.**

Thereare following types of uninformed searches:

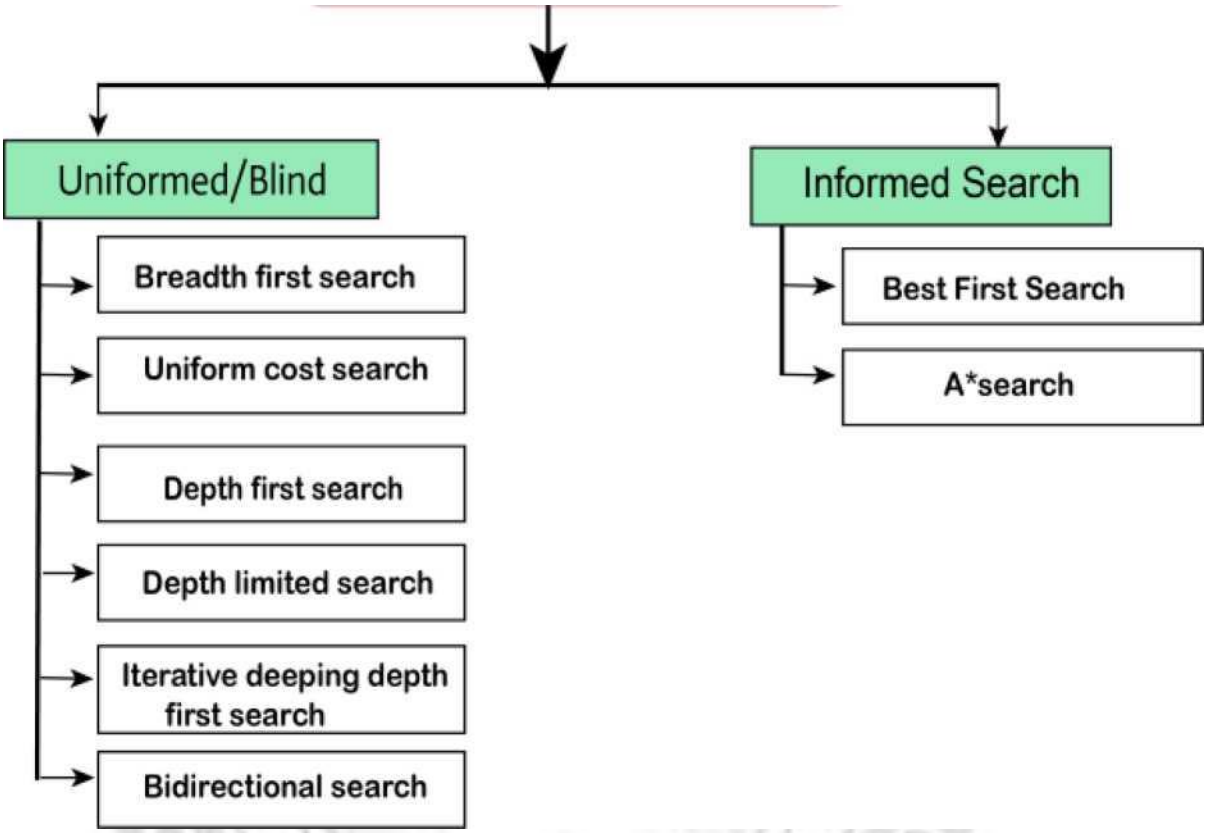
- Breadth-first search
- Uniform cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search
- Bidirectional search

2. ***Informed Search (Heuristic Search)***

This type of search strategy contains some additional information about the states beyond the problem definition. This search uses problem-specific knowledge to find more efficient solutions. This search maintains some sort of internal states via heuristic functions (which provides hints), so it is also called **heuristic search.**

There are following types of informed searches:

- Best first search (Greedy search)
- A* search



Search Algorithm Terminologies:

- **Search:** Searchingis 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.
- **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.
- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.

- **Optimal Solution:** If a solution has the lowest cost among all solutions.

Properties of Search Algorithms:

Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

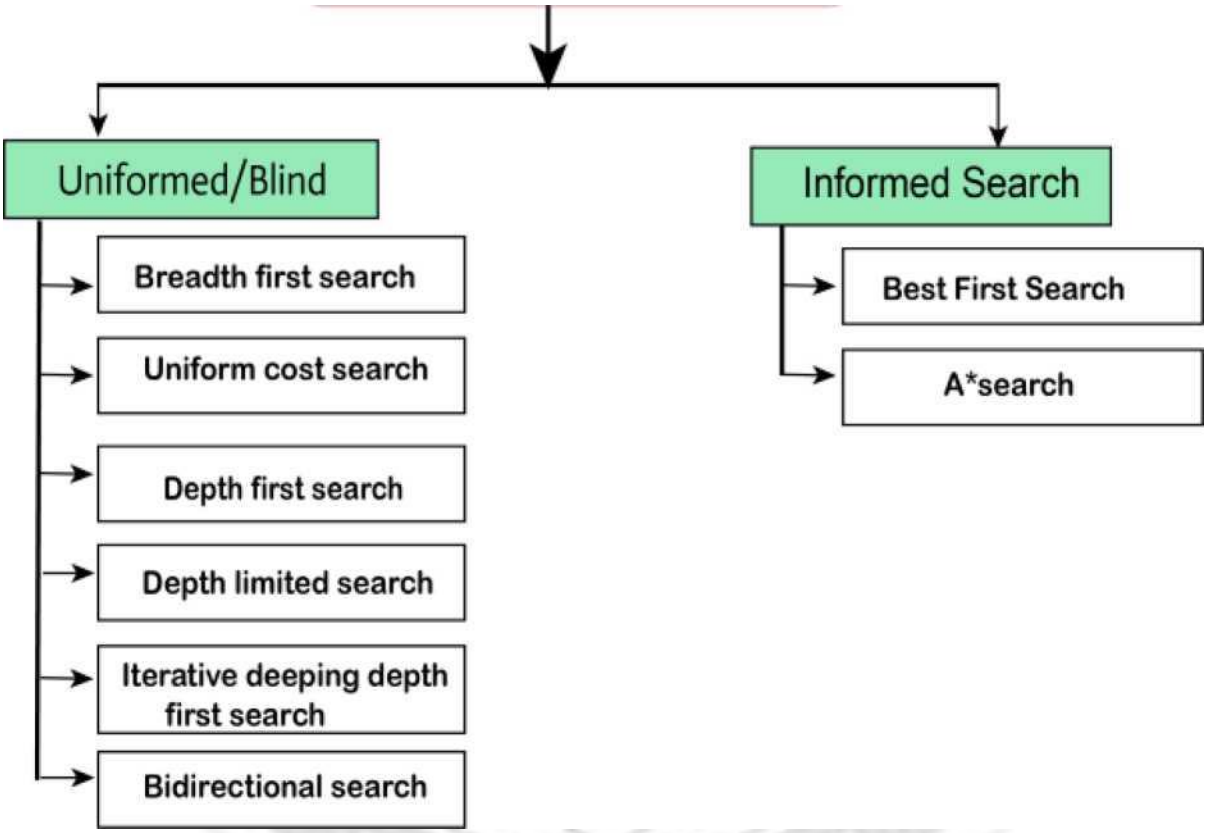
Completeness: A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.

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.

Search Algorithm



Search Algorithm Terminologies:

- **Search:** Searchingis 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.
- **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.
- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.

- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.
- **Optimal Solution:** If a solution has the lowest cost among all solutions.

Properties of Search Algorithms:

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.