# Hill Climbing Algorithm in Artificial Intelligence

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems.
- One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

## Features of Hill Climbing:

Following are some main features of Hill Climbing Algorithm:

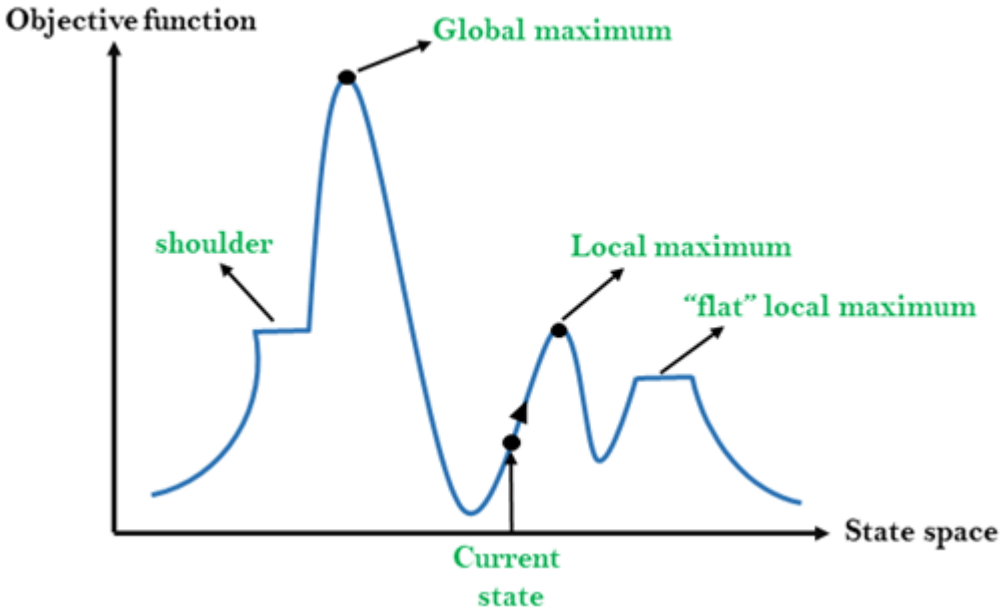
- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the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

## State-space Diagram for Hill Climbing:

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

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

of the search is to find the global maximum and local maximum.




## Different regions in the state space landscape:

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

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

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

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

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

## Types of Hill Climbing Algorithm:

- Simple hill Climbing:
- Steepest-Ascent hill-climbing:

- Stochastic hill Climbing:

## 1. Simple Hill Climbing:

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

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

### Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
    a. If it is goal state, then return success and quit.
    b. Else if it is better than the current state then assign new state as a current state.
    c. Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

## 2. Steepest-Ascent hill climbing:

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

### Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state

as initial state.

- **Step 2:** Loop until a solution is found or the current state does not change.
- **Step 3:** Let SUCC be a state such that any successor of the current state will be better than it.
- **Step4 :** For each operator that applies to the current state:
    a. Apply the new operator and generate a new state.
    b. Evaluate the new state.
    c. If it is goal state, then return it and quit, else compare it to the SUCC.
    d. If it is better than SUCC, then set new state as SUCC.
    e. If the SUCC is better than the current state, then set current state to SUCC.
- **Step 5:** Exit.

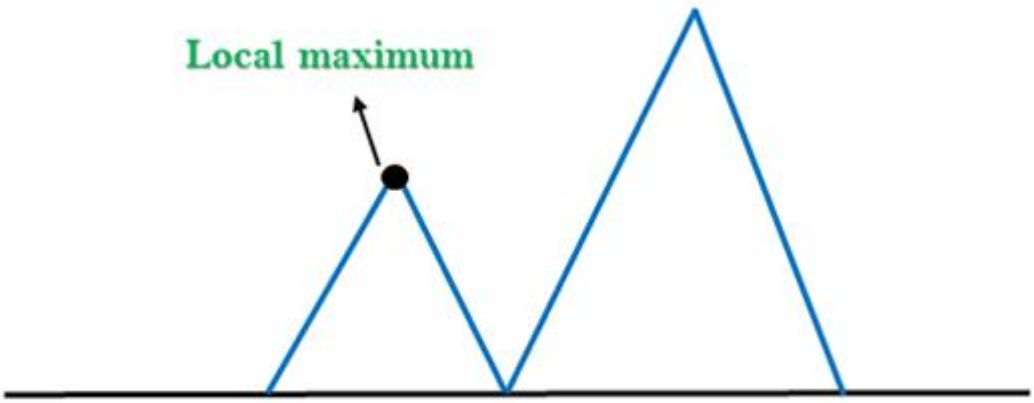
## 3. Stochastic hill climbing:

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

# Problems in Hill Climbing Algorithm:

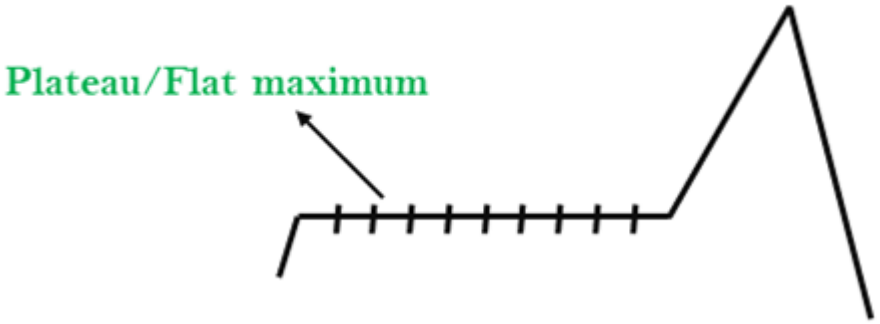
1. **Local Maximum:** A local maximum is a peak state in the landscape which is better than
2. each of its neighboring states, but there is another state also present which is higher than
3. the local maximum.

**Solution:** Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.




**2. Plateau:** A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

**Solution:** The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.




**3. Ridges:** A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.

**Solution:** With the use of bidirectional search, or by moving in different directions, we can improve

this problem.


## Simulated Annealing:

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

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

---

# Best First Search Algorithm in AI | Concept, Implementation, Advantages, Disadvantages

The best first search uses the concept of a priority queue and heuristic search. It is a search algorithm that works on a specific rule. The aim is to reach the goal from the initial state via the shortest path. The best First Search algorithm in artificial intelligence is used for for finding the shortest path from a given starting node to a goal node in a graph. The algorithm works by expanding the nodes of the graph in order of increasing the distance from the starting node until the goal node is reached.

**Search algorithms** form the core of such Artificial Intelligence programs. And while we may be inclined to think that this has limited applicability only in areas of gaming and puzzle-solving, such algorithms are in fact used in many more AI areas like route and cost optimizations, action planning, knowledge mining, robotics, autonomous driving, computational biology, software and hardware verification, theorem proving etc. In a way, many AI problems can be modelled as a search problem where the task is to reach the goal from the initial state via state transformation rules.

All search methods can be broadly classified into two categories:

1. **Uninformed (or Exhaustive or Blind) methods**, where the search is carried out without any additional information that is already provided in the problem statement. Some examples include Breadth-First Search, Depth First Search etc.
2. Informed (or Heuristic) methods, where the search is carried out by using additional information to determine the next step towards finding the solution. BFS is an example of such algorithms

Informed search methods are more efficient, low in cost and high in performance as compared to uninformed search methods.

## What is Best First Search?

If we consider searching as a form of traversal in a graph, an uninformed search algorithm would blindly traverse to the next node in a given manner without considering the cost associated with that step. An informed search, like BFS, on the other hand,

would use an evaluation function to decide which among the various available nodes is the most promising (or 'BEST') before traversing to that node.

BFS uses the concept of a Priority queue and heuristic search. To search the graph space, the BFS method uses two lists for tracking the traversal. An 'Open' list that keeps track of the current 'immediate' nodes available for traversal and a 'CLOSED' list that keeps track of the nodes already traversed.

## Best First Search Algorithm

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the 'ordered' OPEN list
3. Repeat the next steps until the GOAL node is reached
    1. If the OPEN list is empty, then EXIT the loop returning 'False'
    2. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also, capture the information of the parent node
    3. If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path
    4. If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list
    5. Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

This algorithm will traverse the shortest path first in the queue. The time complexity of the algorithm is given by O(n*logn).

## Variants of Best First Search

The two variants of BFS are **Greedy Best First Search** and **A* Best First Search**. Greedy BFS makes use of the Heuristic function and search and allows us to take advantage of both algorithms.

There are various ways to identify the 'BEST' node for traversal and accordingly there are various flavours of BFS algorithm with different heuristic evaluation functions f(n). We will cover the two most popular versions of the algorithm in this blog, namely Greedy Best First Search and A* Best First Search.

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

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

which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

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

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

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

## Greedy Best first search algorithm:

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

## Advantages:

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

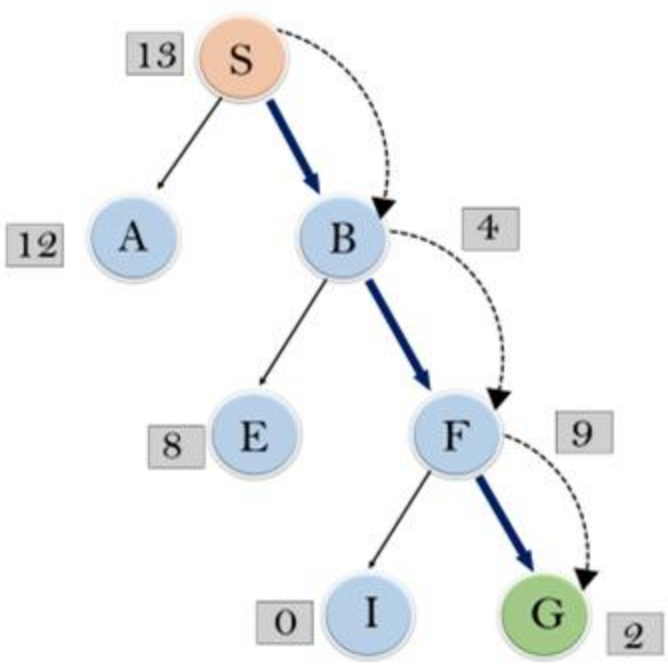
## Disadvantages:

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

## Example:

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

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




**Expand the nodes of S and put in the CLOSED list**

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

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

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

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

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

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

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

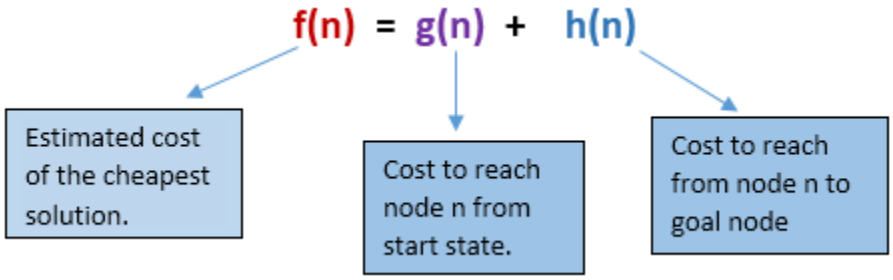
**Complete:** Greedy best-first search is also incomplete, even if the given state space is finite.

**Optimal:** Greedy best first search algorithm is not optimal.

# 2.) A* Search Algorithm:

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

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




## Algorithm of A* search:

**Step1:** Place the starting node in the OPEN list.

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

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

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

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

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

## Advantages:

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

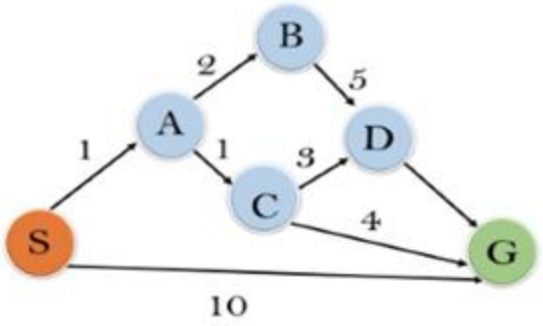
## Disadvantages:

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

## Example:

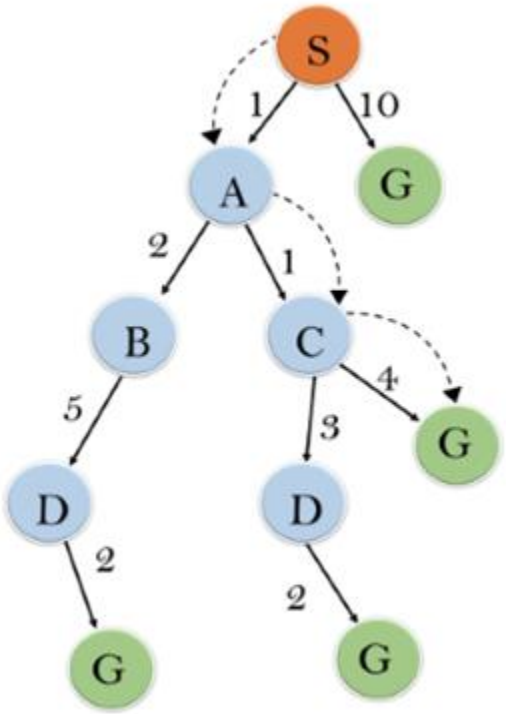
In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.

Here we will use OPEN and CLOSED list.




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

**Solution:**



**Initialization:** {(S, 5)}

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

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

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

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

**Points to remember:**

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

**Complete:** A* algorithm is complete as long as:

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

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

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

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

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

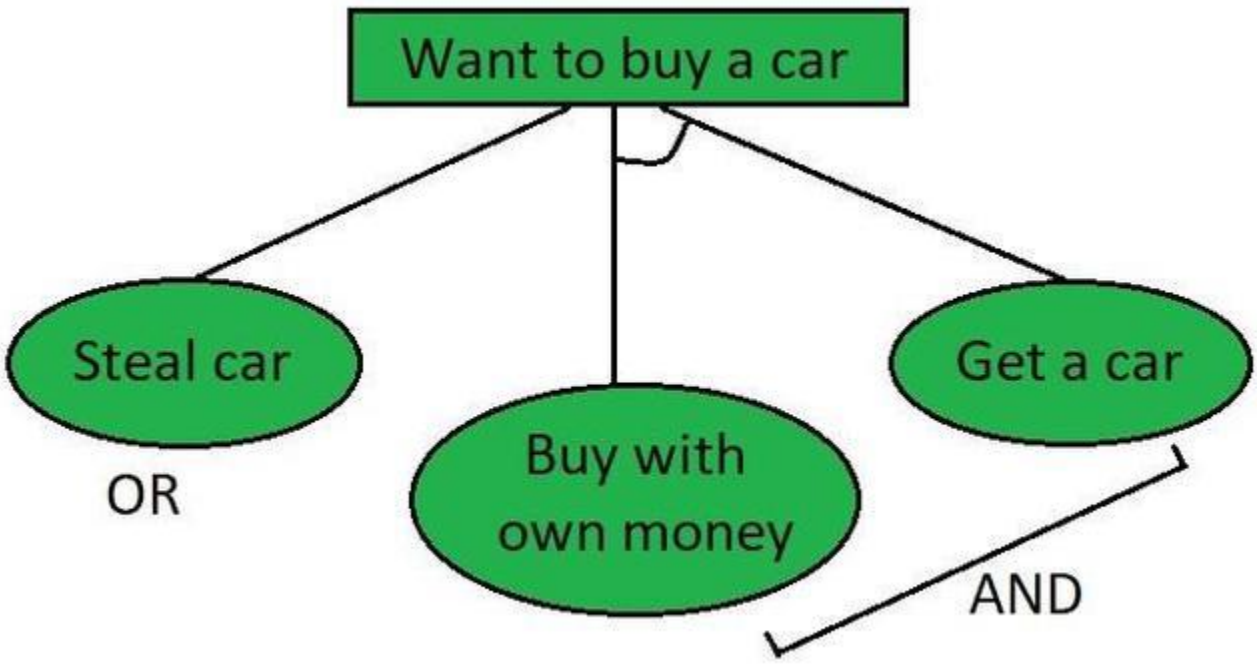
**Space Complexity:** The space complexity of A* search algorithm is **O(b^d)**

## What is problem-reduction ?

*Problem reduction is an algorithm design technique that takes a complex problem and reduces it to a simpler one. The simpler problem is then solved and the solution of the simpler problem is then transformed to the solution of the original problem.*

Problem reduction is a powerful technique that can be used to simplify complex problems and make them easier to solve. It can also be used to reduce the time and space complexity of algorithms.

The AO* method **divides** any given difficult **problem into a smaller group** of problems that are then resolved **using the AND-OR** graph concept. AND OR graphs are specialized graphs that are used in problems that can be divided into smaller problems. The AND side of the graph represents a set of tasks that must be completed to achieve the main goal, while the OR side of the graph represents different methods for accomplishing the same main goal.




*AND-OR Graph*

In the above figure, the **buying of a car** may be broken down into smaller problems or tasks that can be accomplished **to achieve the main goal** in the above figure, which is an example of a simple AND-OR graph. The other task is to either steal a car that will help us accomplish the main goal or use your own money to purchase a car that will accomplish the main goal. The AND symbol is used to indicate the AND part of the graphs, which refers to the need that all subproblems containing the AND to be resolved before the preceding node or issue may be finished.

The start state and the target state are already known in the knowledge-based search strategy known as the **AO* algorithm**, and the best path is identified by heuristics. The informed search technique considerably reduces the algorithm's **time complexity**. The AO* algorithm is far more effective in searching AND-OR trees **than** the A* algorithm.

**Working of AO* algorithm:**

The evaluation function in AO* looks like this:

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

**f(n) = Actual cost + Estimated cost**

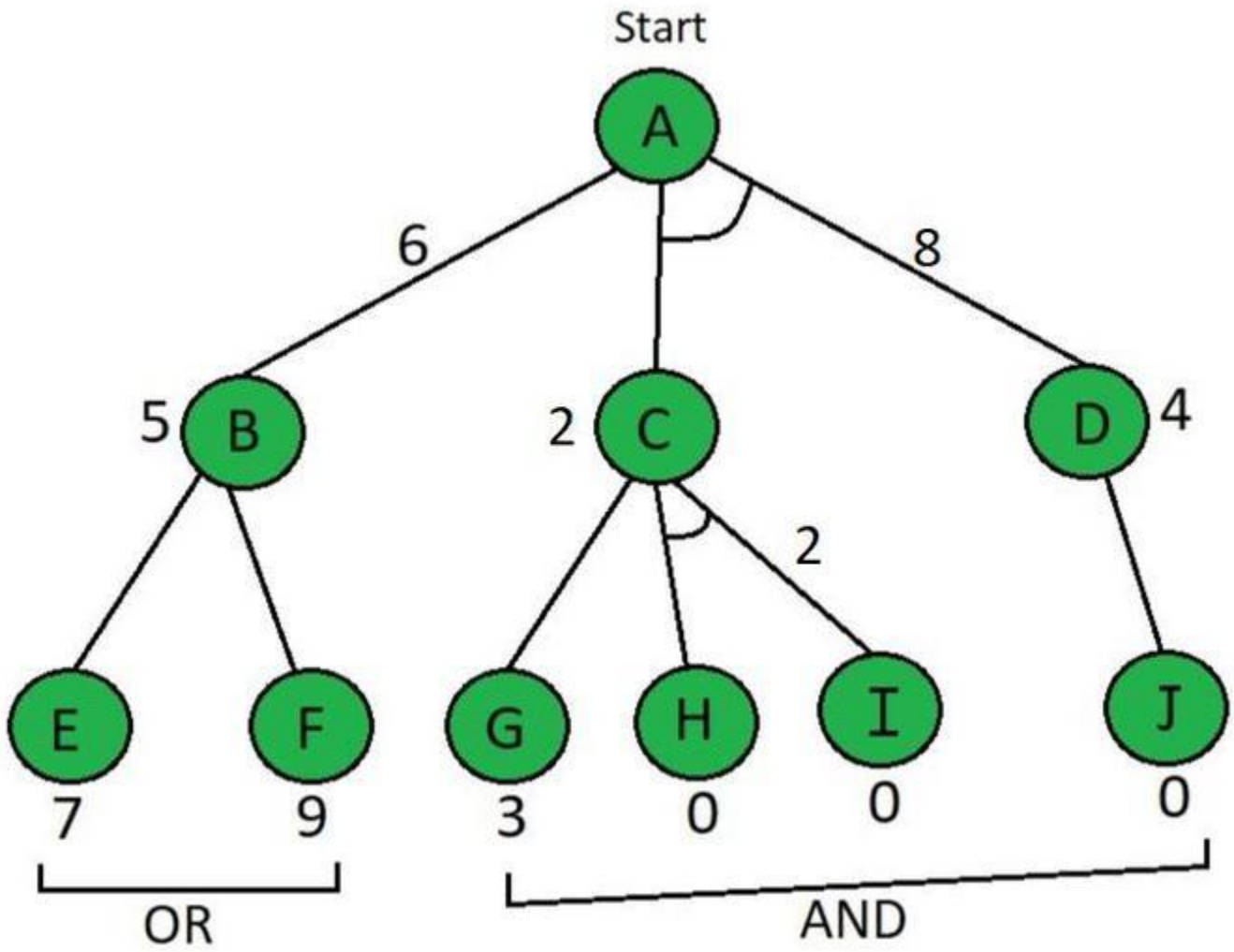
here,

f(n) = The actual cost of traversal.
g(n) = the cost from the initial node to the current node.
h(n) = estimated cost from the current node to the goal state.

**Difference between the A* Algorithm and AO* algorithm**

- A* algorithm and AO* algorithm both works on the **best first search**.
- They are both **informed search** and works on given heuristics values.
- **A*** always **gives** the **optimal solution** but AO* doesn't guarantee to give the optimal solution.
- Once AO* got a solution **doesn't explore** all possible paths but A* explores all paths.
- When compared to the A* algorithm, the AO* algorithm uses **less memory.**
- opposite to the A* algorithm, the AO* algorithm cannot go into an endless **loop.**

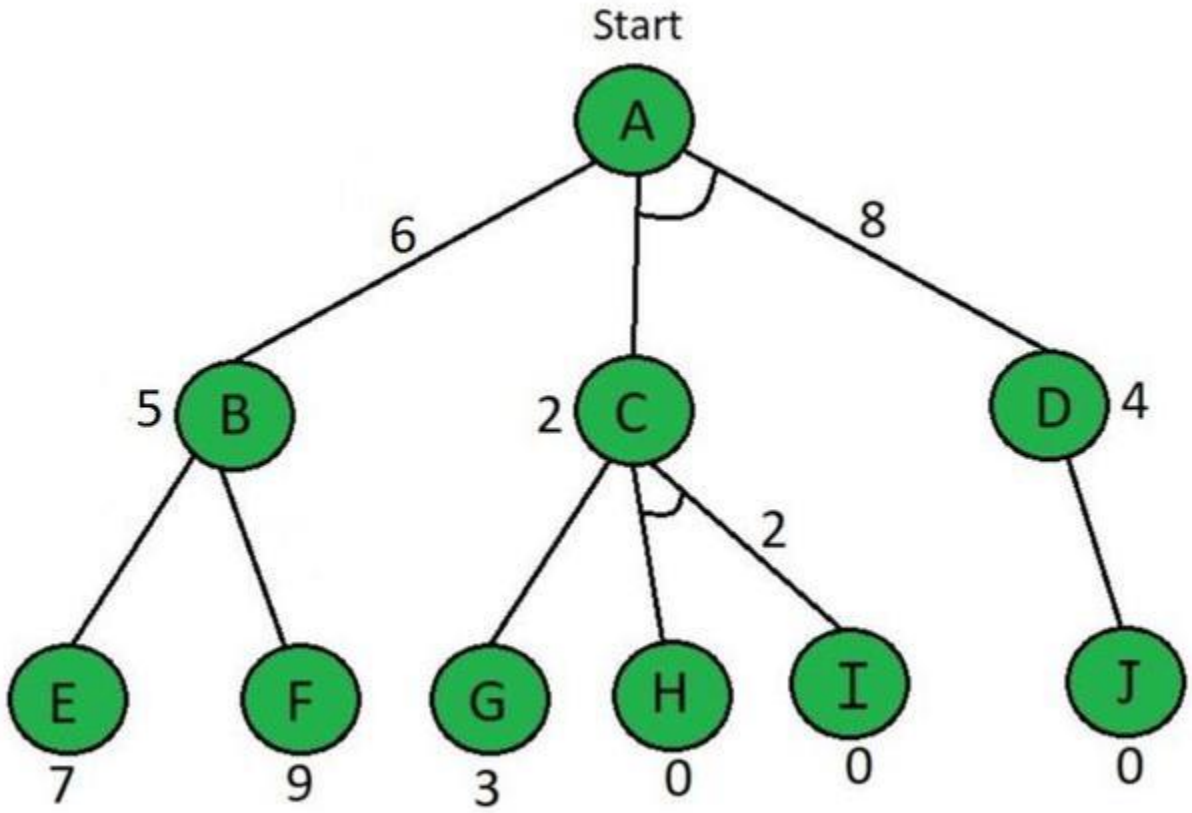
**Example:**




*AO* Algorithm – Question tree*

Here in the above example below the Node which is given is the heuristic value i.e **h(n)**. Edge length is considered as **1**.

## Step 1




*AO* Algorithm (Step-1)*

With help of **f(n) = g(n) + h(n)** evaluation function,
Start from node A,

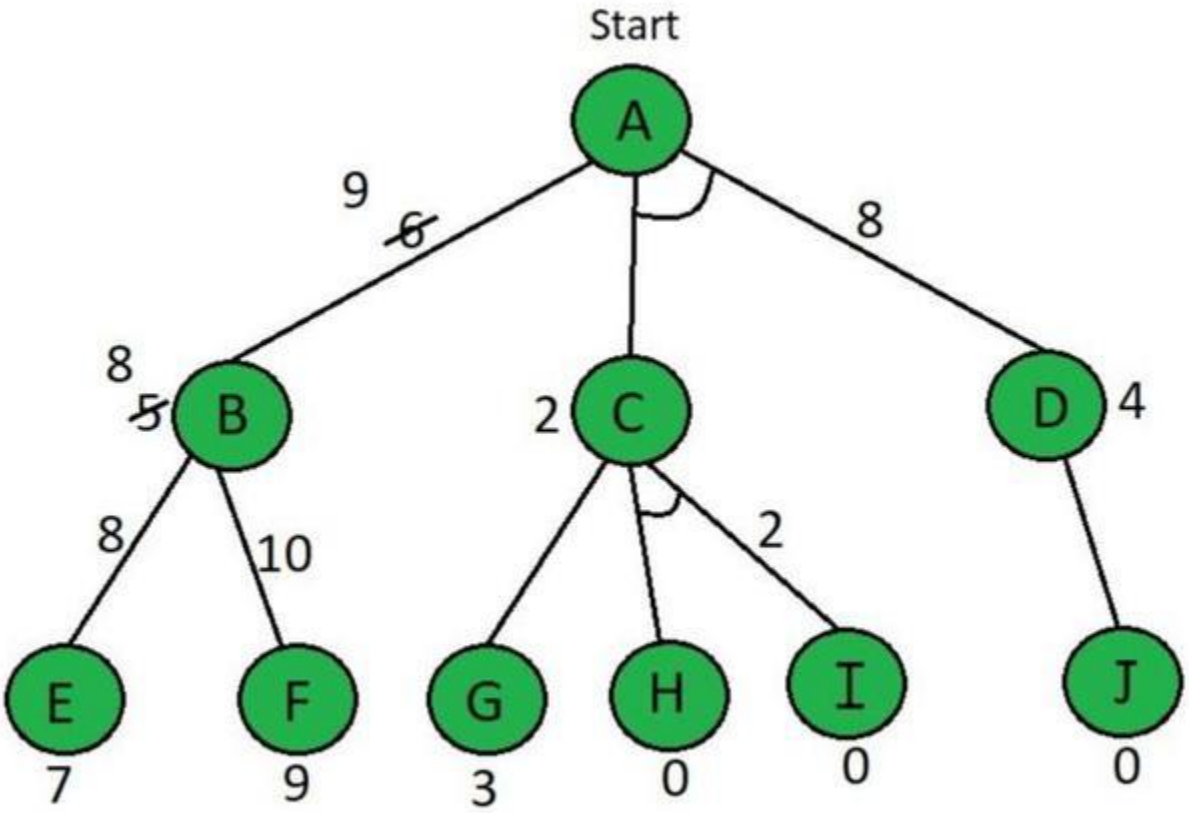
f(A⇢B) = g(B) + h(B)

= 1 + 5 ……here **g(n)=1** is taken by default for path cost
= 6

f(A⇢C+D) = g(c) + h(c) + g(d) + h(d)
= 1 + 2 + 1 + 4 ……here we have added **C & D** because they are in **AND**
= 8

So, by calculation **A⇢B** path is chosen which is the minimum path, i.e **f(A⇢B)**

## Step 2




**AO* Search Procedure.**

**1. Place the start node on open.**

**2. Using the search tree, compute the most promising solution tree TP .**

**3. Select node n that is both on open and a part of tp, remove n from open and place it no closed.**

**4. If n is a goal node, label n as solved. If the start node is solved, exit with success where tp is the solution tree, remove all nodes from open with a solved ancestor.**

**5. If n is not solvable node, label n as unsolvable. If the start node is labeled as unsolvable, exit with failure. Remove all nodes from open ,with unsolvable ancestors.**

**6. Otherwise, expand node n generating all of its successor compute the cost of for each newly generated node and place all such nodes on open.**

**7. Go back to step(2)**

**Must Remember points about Problem Reduction:**

- Reducing a problem to another one is only practical when the total time taken for transforming and solving the reduced problem is lower than solving the original problem.
- If problem A is reduced to problem B, then the lower bound of B can be higher than the lower bound of A, but it can never be lower than the lower bound of A.

# Constraint Satisfaction Problems in Artificial Intelligence

Whenever a problem is actually variables comply with stringent conditions of principles, it is said to have been addressed using the solving multi - objective method. Wow what a method results in a study sought to achieve of the intricacy and organization of both the issue.

Three factors affect restriction compliance, particularly regarding:

- It refers to a group of parameters, or X.
- D: The variables are contained within a collection several domain. Every variables has a distinct scope.
- C: It is a set of restrictions that the collection of parameters must abide by.

In constraint satisfaction, domains are the areas wherein parameters were located after the restrictions that are particular to the task. Those three components make up a constraint satisfaction technique in its entirety. The pair "scope, rel" makes up the number of something like the requirement. The scope is a tuple of variables that contribute to the restriction, as well as rel is indeed a relationship that contains a list of possible solutions for the parameters should assume in order to meet the restrictions of something like the issue.

Issues with Contains A certain amount Solved

For a constraint satisfaction problem (CSP), the following conditions must be met:

- States area
- fundamental idea while behind remedy.

The definition of a state in phase space involves giving values to any or all of the parameters, like as

X1 = v1, X2 = v2, etc.

There are 3 methods to economically beneficial to something like a parameter:

1. Consistent or Legal Assignment: A task is referred to as consistent or legal if it complies with all laws and regulations.
2. Complete Assignment: An assignment in which each variable has a number associated to it and that the CSP solution is continuous. One such task is referred to as a completed task.
3. A partial assignment is one that just gives some of the variables values. Projects of this nature are referred to as incomplete assignment.

## Domain Categories within CSP

The parameters utilize one of the two types of domains listed below:

- Discrete Domain: This limitless area allows for the existence of a single state with numerous variables. For instance, every parameter may receive a endless number of beginning states.
- It is a finite domain with continous phases that really can describe just one area for just one particular variable. Another name for it is constant area.

## Types of Constraints in CSP

Basically, there are three different categories of limitations in regard towards the parameters:

- Unary restrictions are the easiest kind of restrictions because they only limit the value of one variable.
- Binary resource limits: These restrictions connect two parameters. A value between x1 and x3 can be found in a variable named x2.
- Global Resource limits: This kind of restriction includes a unrestricted amount of variables.

The main kinds of restrictions are resolved using certain kinds of resolution methodologies:

- In linear programming, when every parameter carrying an integer value only occurs in linear equation, linear constraints are frequently utilised.
- Non-linear Constraints: With non-linear programming, when each variable (an integer value) exists in a non-linear form, several types of restrictions were utilised.

**EXAMPLES->**

Sudoku puzzle where some of the squares have initial fills of certain integers.

You must complete the empty squares with numbers between 1 and 9, making sure that no rows, columns, or blocks contains a recurring integer of any kind. This solving multi - objective issue is pretty elementary. A problem must be solved while taking certain limitations into consideration.

The integer range (1-9) that really can occupy the other spaces is referred to as a domain, while the empty spaces themselves were referred as variables. The values of the variables are drawn first from realm. Constraints are the rules that determine how a variable will select the scope.