UNIT 2

What is Environment and its features

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.

- Fully observable vs Partially Observable
- Static vs Dynamic
- Discrete vs Continuous
- Deterministic vs Stochastic
- Single-agent vs Multi-agent
- Episodic vs sequential
- Known vs Unknown
- Accessible vs Inaccessible

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.

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.

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.

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.

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.

Discrete vs Continuous:

If in an environment there are a finite number of percepts and actions that can be performed within it, then such an environment is called a discrete environment else it is called continuous environment. A chess gamecomes under discrete environment as there is a finite number of moves that can be performed. A self-driving car is an example of a continuous environment.

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.

Accessible vs Inaccessible:

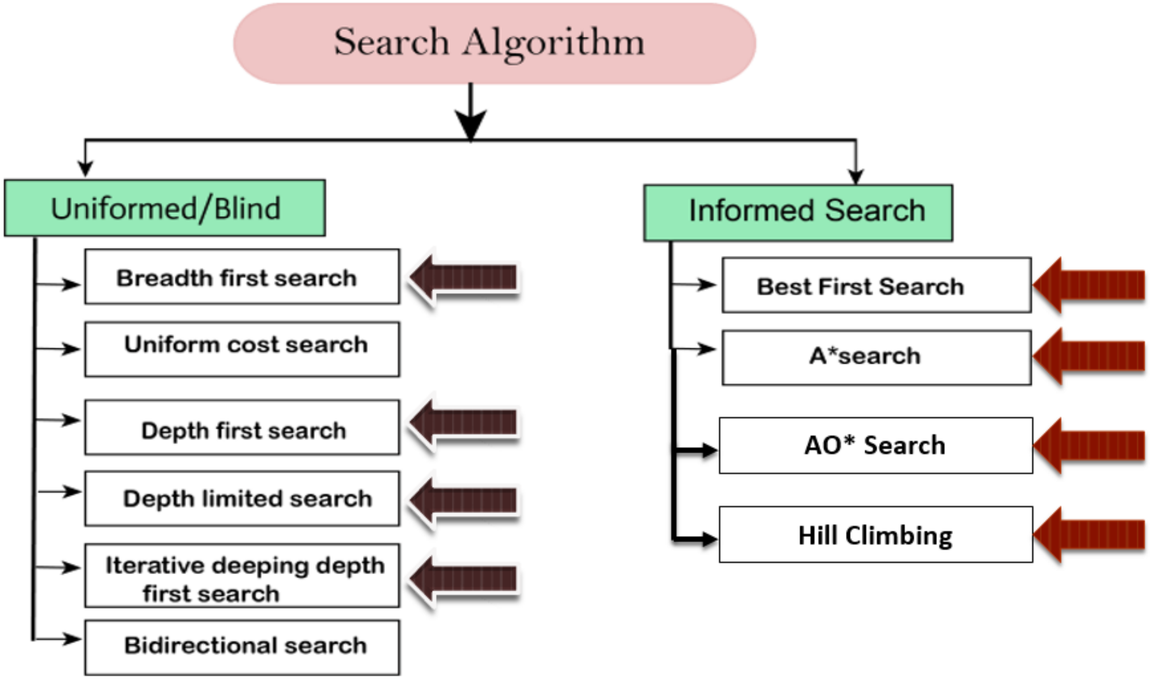
If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible. An empty room whose state can be defined by its temperature is an example of an accessible environment. Information about an event on earth is an example of Inaccessible environment.

What is Searching?

Search algorithms are one of the most important areas of Artificial Intelligence. This topic will explain all about the search algorithms in AI. Problem-solving agents: In Artificial Intelligence, Search techniques are universal problem-solving methods.

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

Search is the fundamental technique of AI. Possible answers, decisions or courses of action are structured into an abstract space, which we then search. Please check below searching algorithms.



Search Algorithm Terminologies

- **Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
- **Search Space:** Search space represents a set of possible solutions, which a system may have.
- **Start State:** It is a state from where agent begins the search.
- **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 Algorithm

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.

Uninformed Search Algorithms

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

Following are the various types of uninformed search algorithms:

1. Breadth-first Search
2. Depth-first Search
3. Depth-limited Search
4. Iterative deepening depth-first search

Breadth-first Search:

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

Advantages:

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

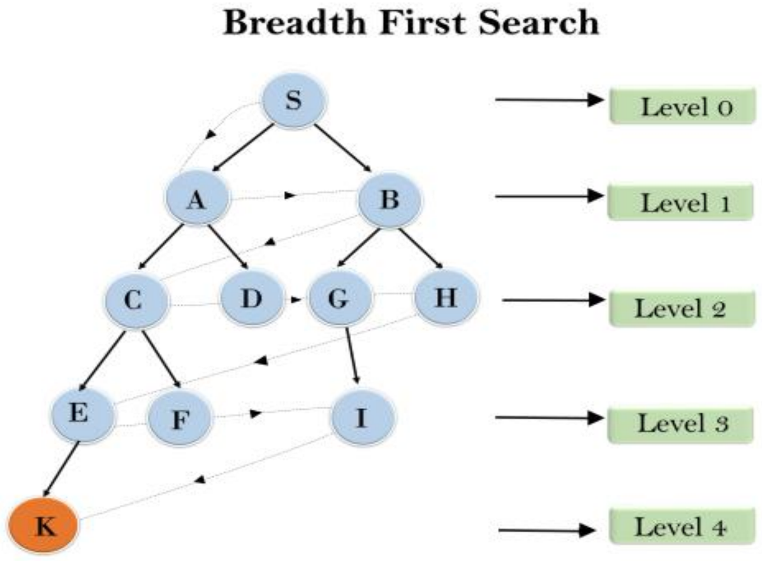
Disadvantages:

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

Example:

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

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



Depth-first Search

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

Advantage:

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

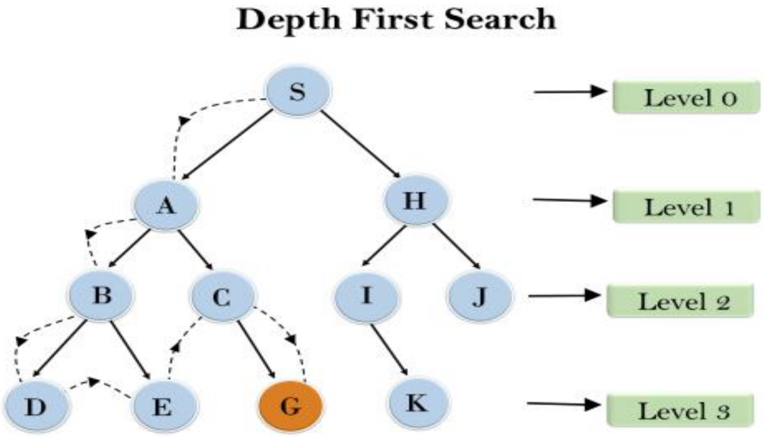
Disadvantage:

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

Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as: Root node--->Left node ----> right node.

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



Depth-Limited Search Algorithm:

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

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

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

Advantages:

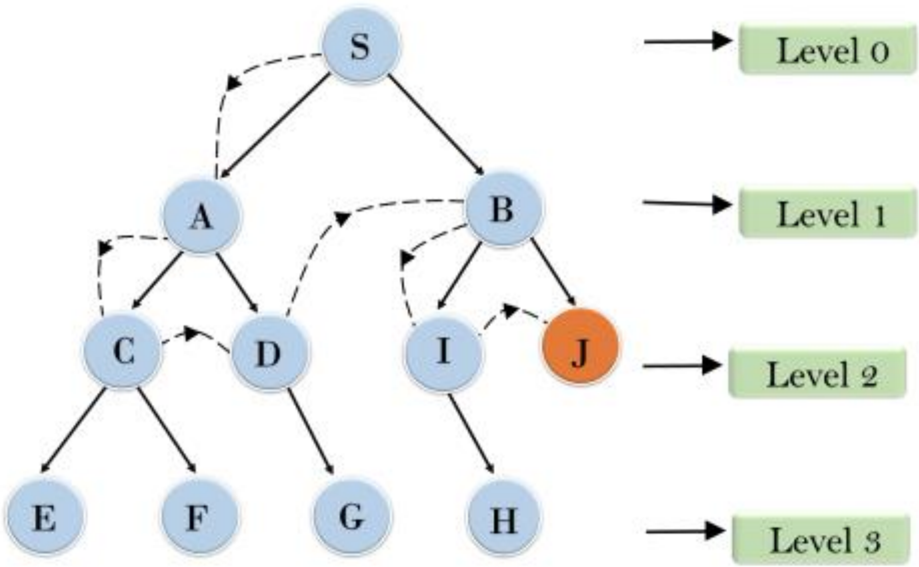
Depth-limited search is Memory efficient.

Disadvantages:

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

Example:

Depth Limited Search



Informed Search

Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.

A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.

Informed search can solve much complex problem which could not be solved in another way.

Informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc.

This knowledge help agents to explore less to the search space and find more efficiently the goal node. An example of informed search algorithms is a Traveling Salesman Problem.

Informed Search: Heuristics function

Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.

The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time.

Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as: $h(n) <= h^*(n)$

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

Pure Heuristic Search

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded. On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.

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

1. A* Search Algorithm
2. AO* Search Algorithm
3. Best First Search Algorithm (Greedy search)
4. Hill Climbing

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 (Uniform Cost Search) 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:

- **Step1:** Place the starting node in the OPEN list.
- **Step 2:** Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
- **Step 3:** Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
- **Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.
- **Step 5:** Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
- **Step 6:** Return to **Step 2**.

Advantages:

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

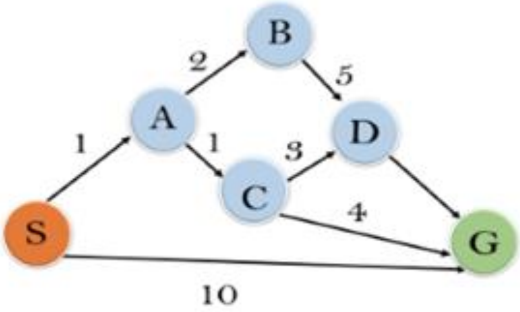
Disadvantages:

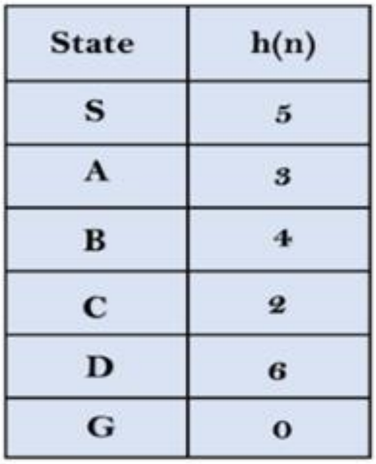
- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example:

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

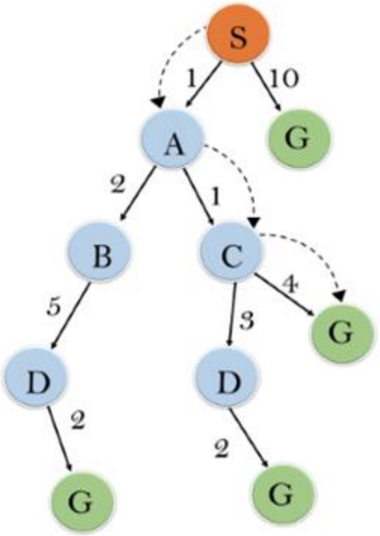
Here we will use OPEN and CLOSED list.





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

Solution:



Initialization: {(S, 5)}

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

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

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

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

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)

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:

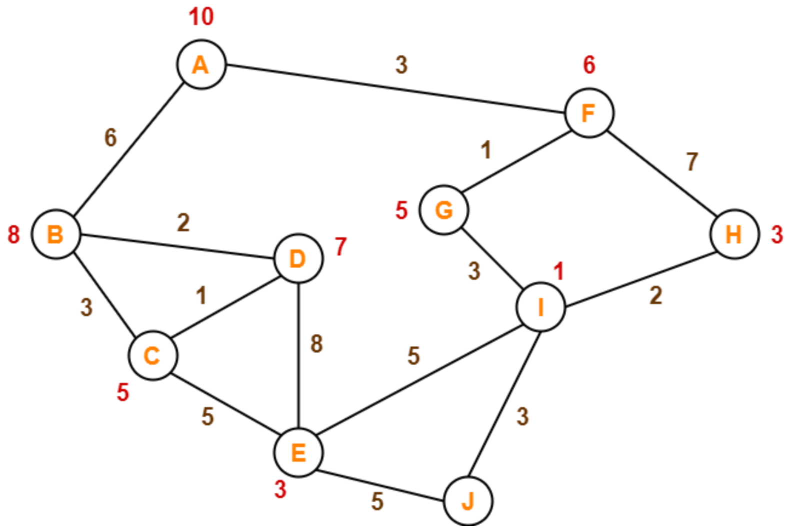
- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.

- **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.

Practice Examples:

Start Node= A **Goal Node= J**



Informed Search: AO* Search Algorithm

AO* search Algorithm is based on problem decomposition (Breakdown problem into small pieces) When a problem can be divided or decomposed into a set of sub problems, where each sub problem can be solved separately and for each sub problem, sub solution is evaluated and a combination of these sub solutions will be a whole solution, AND OR graphs or AND OR trees are used for representing this solution.

Problem Reduction in Artificial Intelligence

AO* is informed search algorithm, work based on heuristic. We already know about the divide and conquer strategy, a solution to a problem can be obtained by decomposing it into smaller sub-problems.

Each of this sub-problem can then be solved to get its sub solution. These sub solutions can then recombined to get a solution as a whole. That is called is Problem Reduction. AND-OR graphs or AND – OR trees are used for representing the solution.

This method generates arc which is called as AND-OR arcs. One AND arc may point to any number of successor nodes, all of which must be solved in order for an arc to point to a solution. AND-OR graph is used to represent various kind of complex problem solutions.
AO* search algorithm is based on AND-OR graph so, it is called AO* search algorithm.

Like A* algorithm here we will use two arrays and one heuristic function.

OPEN: It contains the nodes that has been traversed but yet not been marked solvable or unsolvable.

CLOSE: It contains the nodes that have already been processed.

Algorithm:

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in CLOSE.

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved. If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable, then return failure and exit.

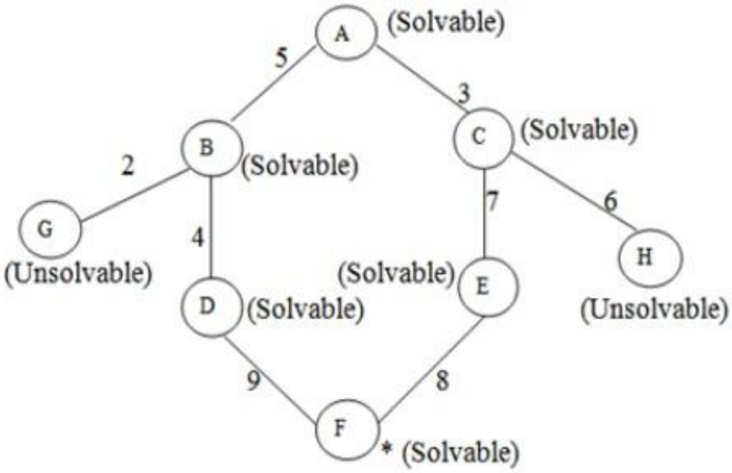
Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPE

Step 7: Return to Step 2.

Step 8: Exit.

Example:

Let us take the following example to implement the AO* algorithm.



Figure

Step 1: In the above graph, the solvable nodes are A, B, C, D, E, F and the unsolvable nodes are G, H. Take A as the starting node. So place A into OPEN.

i.e. OPEN = $\boxed{A}$ CLOSE = (NULL) $\boxed{\Phi}$

Step 2:

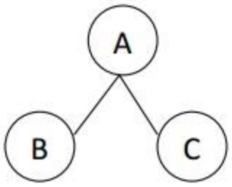
The children of A are B and C which are solvable. So place them into OPEN and place A into the CLOSE.

i.e. OPEN =

B	C

CLOSE =

A



Step 3:

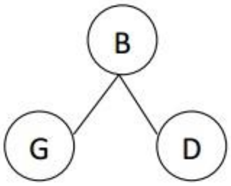
Now process the nodes B and C. The children of B and C are to be placed into OPEN. Also remove B and C from OPEN and place them into CLOSE.

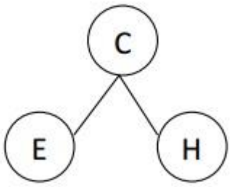
So OPEN =

G	D	E	

C

A	B	C





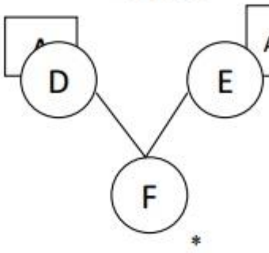
(O)

'O' indicated that the nodes G and H are unsolvable.

Step 4:

As the nodes G and H are unsolvable, so place them into CLOSE directly and process the nodes D and E.

i.e. OPEN = CLOSE =



A	B	C		$G^{(O)}$	D	E		$H^{(O)}$

Step 5:

Now we have been reached at our goal state. So place F into CLOSE.

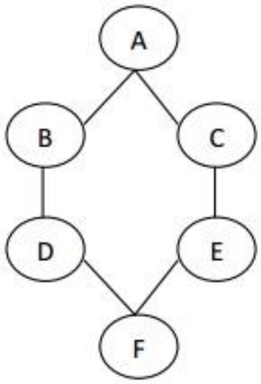
A	B	C		$G^{(O)}$	D	E		$H^{(O)}$	F

i.e. CLOSE =

Step 6:

Success and Exit

AO* Graph:



Figure

Advantages:

- It is an optimal algorithm.
- If traverse according to the ordering of nodes. It can be used for both OR and AND graph.

Disadvantages:

- Sometimes for unsolvable nodes, it can’t find the optimal path. Its complexity is than other algorithms.

Informed Search: Greedy best-first search

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search.

Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e. f(n)= g(n).

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

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

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.

Informed Search: Hill Climbing

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.

Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.

It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.

A node of hill climbing algorithm has two components which are state and value. Hill Climbing is mostly used when a good heuristic is available. In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

Features of Hill Climbing:

Following are some main features of Hill Climbing Algorithm:

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

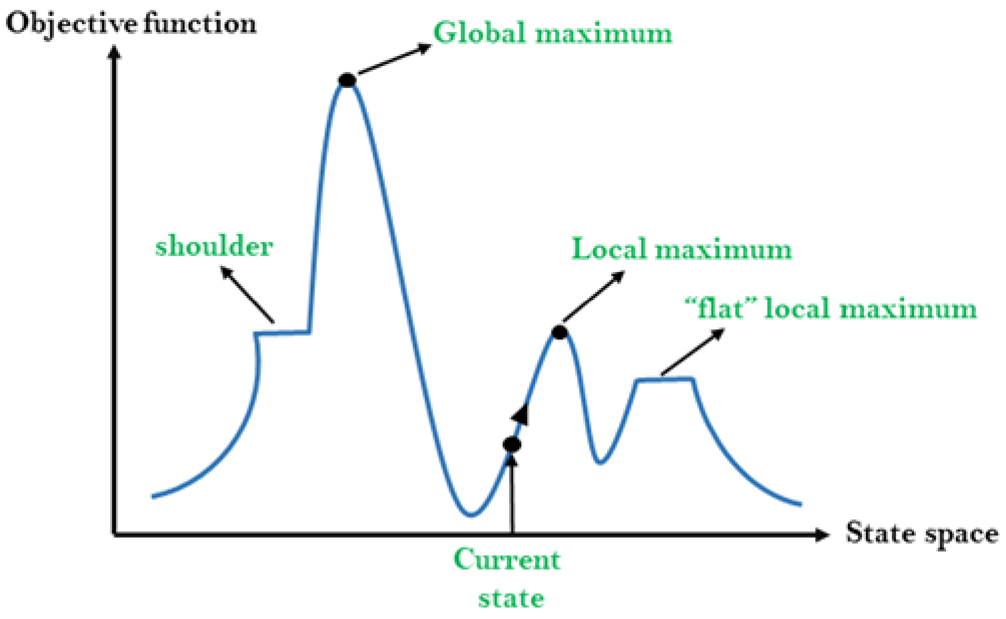
State-space Diagram for Hill Climbing:

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

On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis.

If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.

Different regions in the state space landscape:



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

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

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

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

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

Types of Hill Climbing Algorithm:

1. Simple hill Climbing:
2. Steepest-Ascent hill-climbing:
3. 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.

A. Let SUCC be a state such that any successor of the current state will be better than it.
B. For each operator that applies to the current state:
    a. Apply the new operator and generate a new state.
    b. Evaluate the new state.

c. If it is goal state, then return it and quit, else compare it to the SUCC.
d. If it is better than SUCC, then set new state as SUCC.
e. If the SUCC is better than the current state, then set current state to SUCC.

Step 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.