ARTIFICIAL INTELLIGENCE

Prof. U. M Rane

Artificial Intelligence & Data Science

K. K. Wagh Institute of Engineering Education & Research Nashik

K. K. WAGH EDUCATION SOCIETY

K K Wagh

UNIT- 2

PROBLEM SOLVING USING SEARCH TECHNIQUES

Problem solving Agents in AI

- In artificial intelligence, a problem-solving agent refers to a type of intelligent agent designed to address and solve complex problems or tasks in its environment. These agents are a fundamental concept in AI and are used in various applications, from game-playing algorithms to robotics and decision-making systems.
- When the correct action to take is not immediately obvious, agent may need to plan ahead to consider a sequence of actions that form a path to a goal state, they try to search the path to get to goal state.
- Initial State: The starting point or the state from which the problem-solving process begins.
- Goal State: The desired outcome or the state the agent aims to achieve.
- Actions: A set of possible moves or steps the agent can take to transition from one state to another.
- Transition Model: Describes the results of taking an action in a given state (often represented as a state transition function).
- Path Cost: A function that assigns a numerical cost to each path, allowing the agent to evaluate and compare different paths.

Problem solving by searching

- Problem-solving agents in AI leverage a combination of problem definition and search strategies to find solutions effectively. Depending on the nature of the problem and the requirements (such as optimality and efficiency), different search strategies are chosen to navigate the problem space and reach the goal state.

- The problem solving approach has been applied to a vast array of task environment, standardized problems and real world problems such as- VACCUM WORLD, SOKOBAN PUZZLE.

- Problem Solving in games such as “Sudoku” can be an example. It can be done by building an artificially intelligent system to solve that particular problem. To do this, one needs to define the problem statements first and then generating the solution by keeping the conditions in mind.

Some of the most popularly used problem solving with the help of artificial intelligence are:

- Vacuum word problem
- Puzzle problem
- Travelling Salesman Problem.
- Water-Jug Problem.

Foundation Flow Diagram for Finding solution

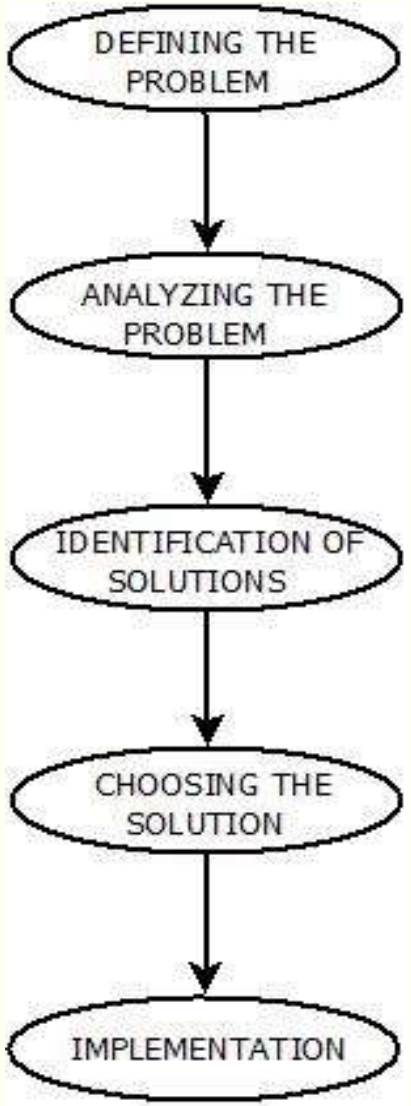
Defining The Problem: The definition of the problem must be included precisely. It should contain the possible initial as well as final situations which should result in acceptable solution.

Analyzing The Problem: Analyzing the problem and its requirement must be done as few features can have immense impact on the resulting solution.

Identification Of Solutions: This phase generates reasonable amount of solutions to the given problem in a particular range.

Choosing a Solution: From all the identified solutions, the best solution is chosen basis on the results produced by respective solutions.

Implementation: After choosing the best solution, its implementation is done.



Example- Vaccum Cleaner

- Initial State- Any state can be assigned as initial State
- Action- In Vacuum each state has 3 actions

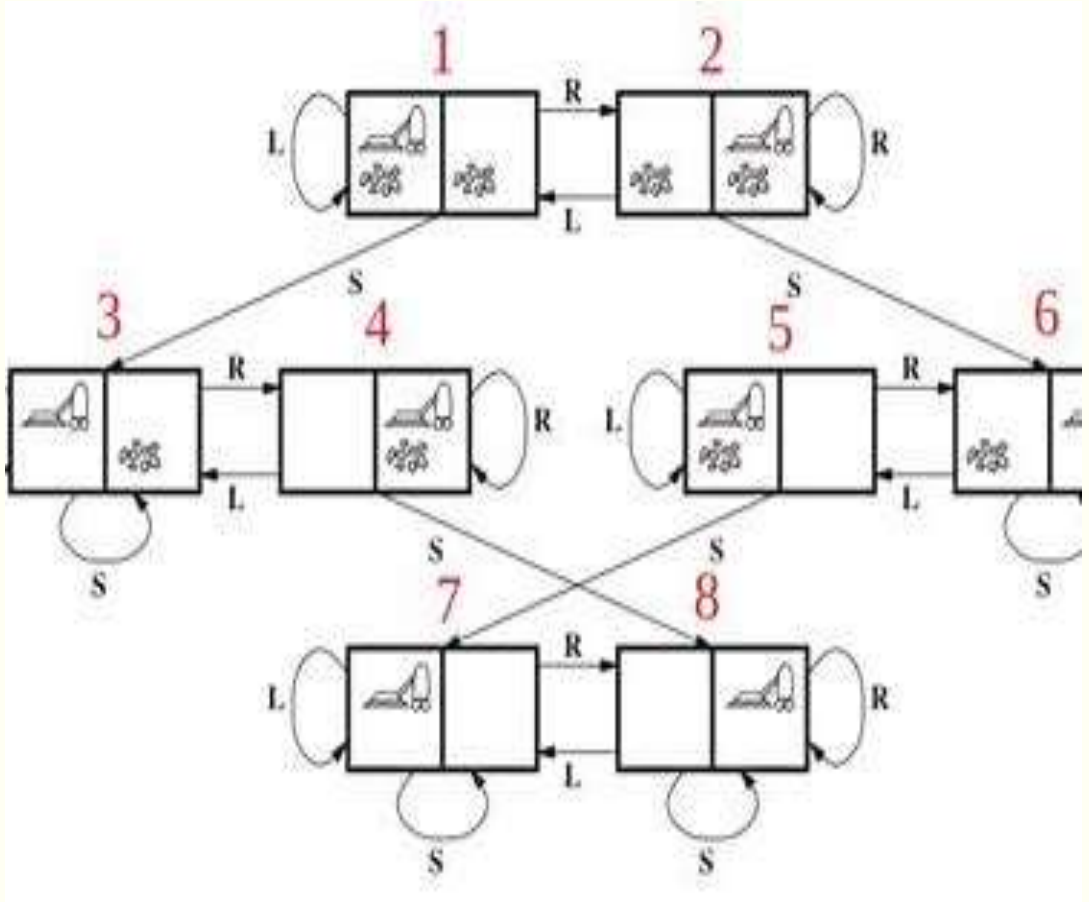
1) Left, Right, clean action

* Successor function- moving left, moving right, clean

*Goal State- Checks whether all the squares are clean

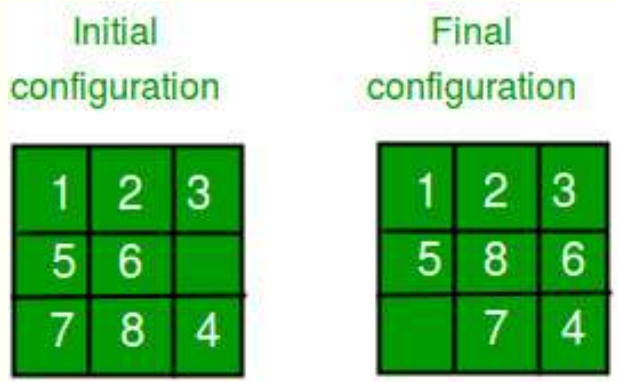
*Path State- each step cost 1 so path cost is no of steps

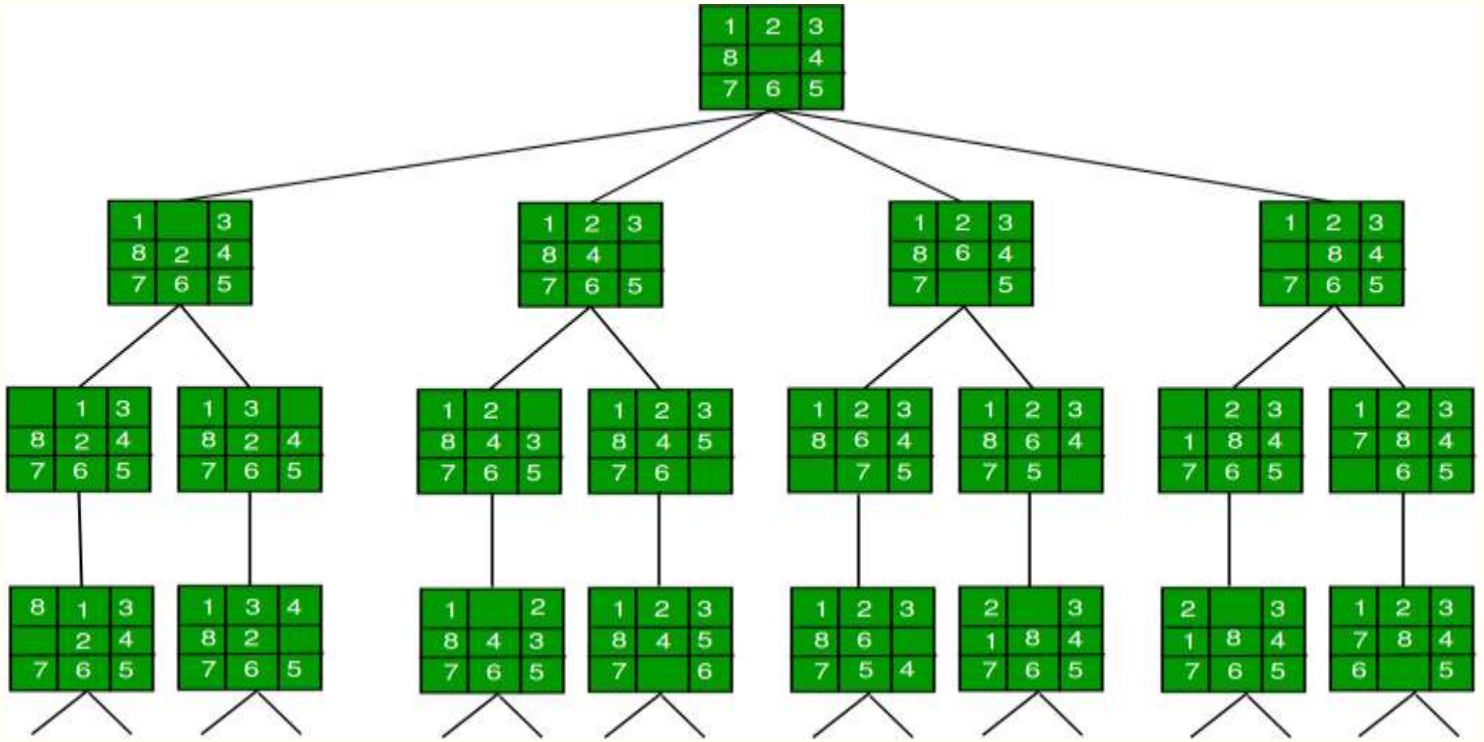
(2x2^n)



Example- 8 Puzzle problem

- The initial state is given to us we need to reach goal state
- Action- either left, right, up, down
- goal state will verify whether we reached our goal state or not.
- path will be number of state formed





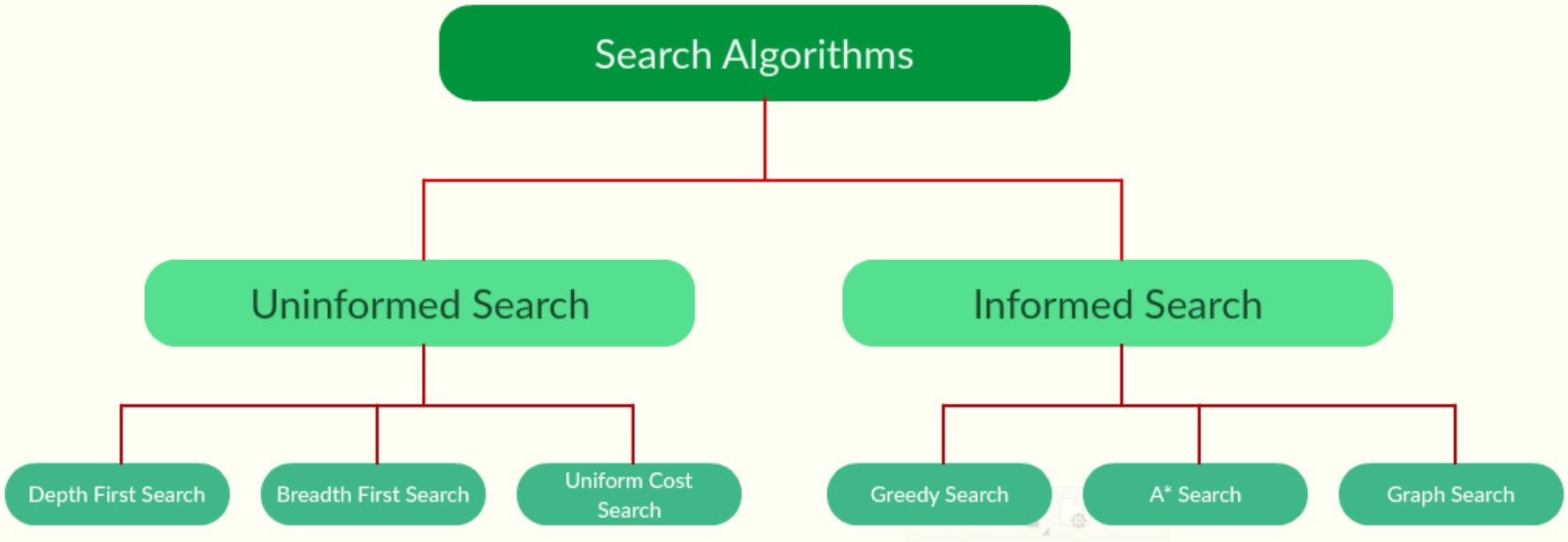
Search Strategies in Artificial Intelligence

K K Wagh

- Uninformed Search (Blind Search): These strategies do not use any domain-specific knowledge beyond the problem definition.
- Breadth-First Search (BFS): Explores the shallowest nodes first. It is complete and optimal for uniform path cost but can be memory-intensive.
- Depth-First Search (DFS): Explores the deepest nodes first. It is less memory-intensive but not guaranteed to be optimal or complete.
- Bidirectional Search: Bidirectional search runs two simultaneous searches: one forward from the initial state and one backward from the goal state. When these two searches meet, the algorithm can construct the shortest path.
- Depth-Limited Search: DFS with a limit on depth, preventing infinite descent.

Uninformed Search Strategies

- The search algorithms in this section have no additional information on the goal node other than the one provided in the problem definition.
- The plans to reach the goal state from the start state differ only by the order and/or length of actions.
- Uninformed search is also called Blind search. These algorithms can only generate the successors and differentiate between the goal state and non goal state.



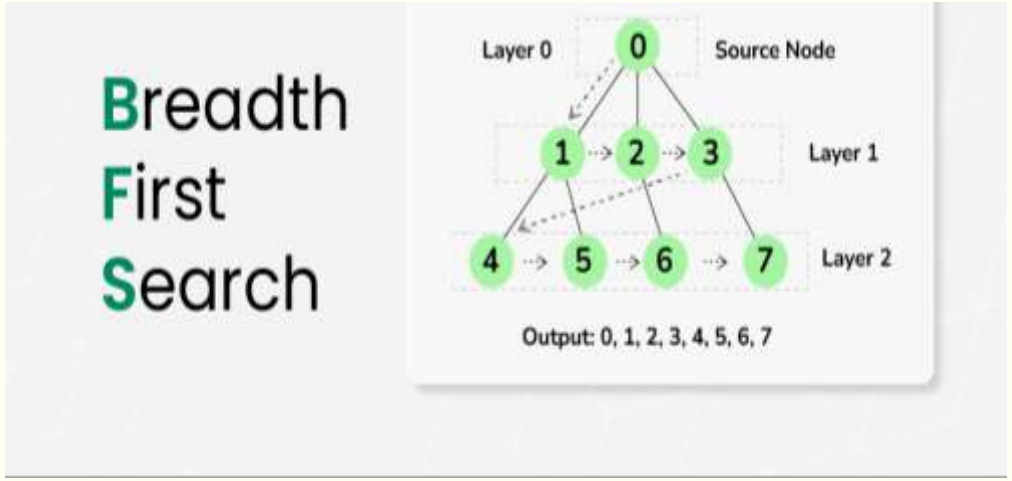
3) Informed Search Strategies

- Informed Search (Heuristic Search): These strategies use heuristics to guide the search process more efficiently.
- Greedy Best-First Search: Expands the node with the lowest heuristic value first, aiming to reach the goal quickly but not guaranteed to be optimal.
- A Search*: Combines the path cost and heuristic value to ensure both optimality and efficiency, provided the heuristic is admissible (never overestimates the cost to reach the goal).
- Heuristic Functions: Used to estimate the cost to reach the goal from a given state (e.g., Manhattan distance, Euclidean distance).
- Here the algorithms have information about the goal state, which helps in more efficient searching. This information is obtained by something called a heuristic.

Uninformed- Breadth First Search Algorithm

- 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.

- Uses Queue(data structure)
- FIFO- first in first out
- Gives Optimal Solution
- Time complexity- O(b^d)
- works Horizontally and uses every possible path on way



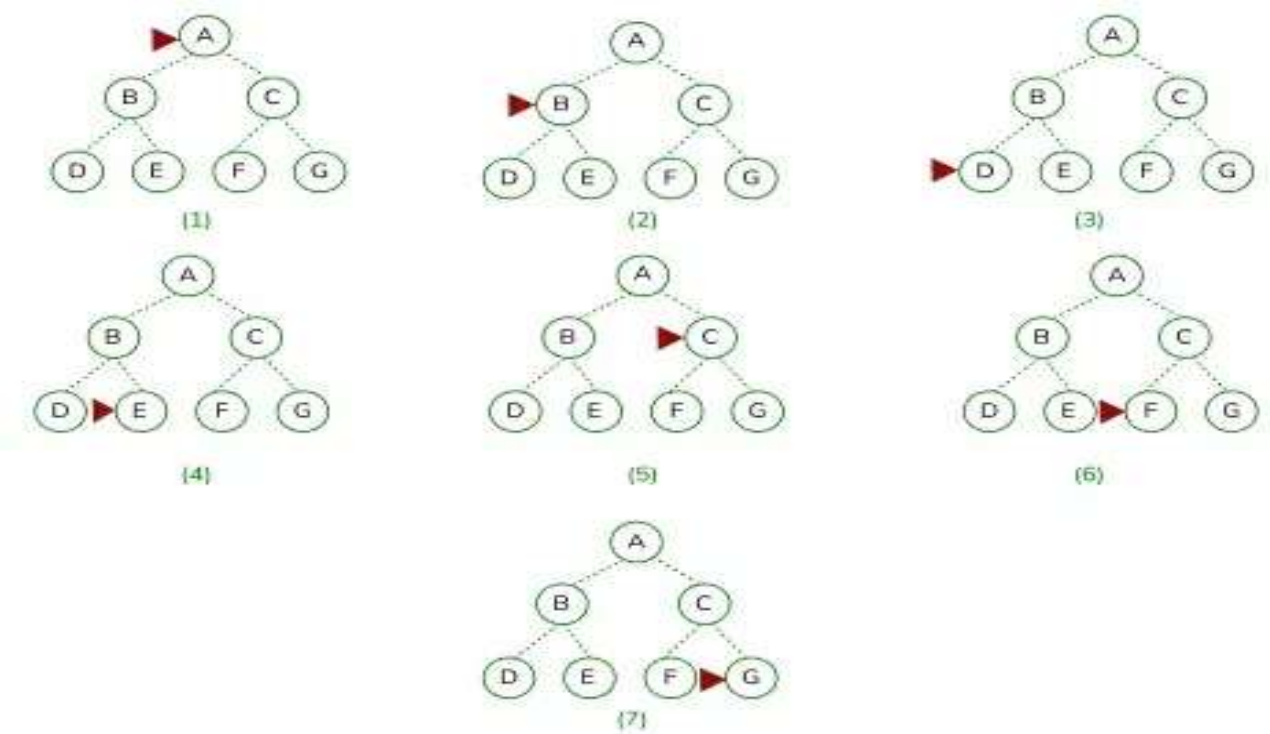
Psudocode BFS

- Enter Starting node in Queue
- if queue is empty return fail stop
- if first element in queue is goal node then return success- stop
- else- Remove first element in queue and place children nodes
- end of queue
- go to step 2

Uninformed- Depth First Search

- Depth-first search is a traversing algorithm used in tree and graph-like data structures.
- It generally starts by exploring the deepest node in the frontier.
- Starting at the root node, the algorithm proceeds to search to the deepest level of the search tree until nodes with no successors are reached.
- Suppose the node with unexpanded successors is encountered then the search backtracks to the next deepest node to explore alternative paths.

- uses LIFO- last in first out
- works vertically or one direction
- Does not Guarantee a optimal sol
- uses stacks(DS)
- Time complexity- O(b^d)



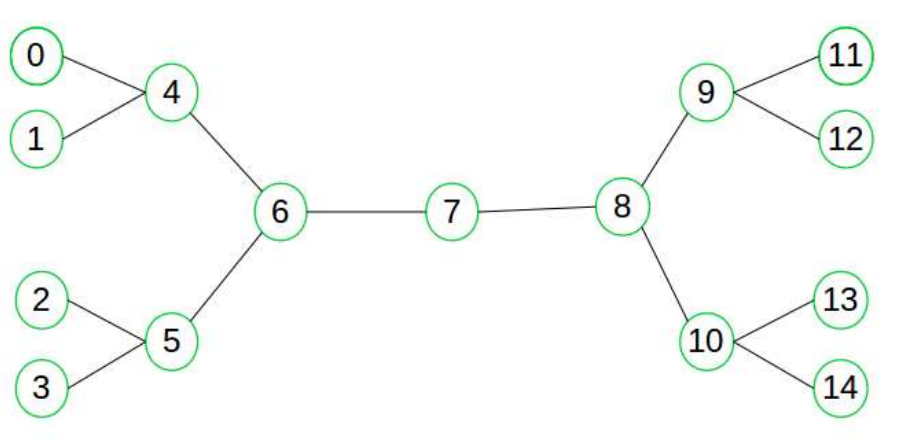
5) Uninformed- Depth Limited Search

- Similar to DFS but set a goal state criteria
- Terminates if solution is found else it stops.
- solves problem of limited depth path
- Does not guarantee the best solution

5) Uninformed- Bidirectional Search

*Bidirectional search is a graph search algorithm which find smallest path from source to goal vertex. It runs two simultaneous search –

Forward search from source/initial vertex toward goal vertex
Backward search from goal/target vertex toward source vertex



*Bidirectional search replaces single search graph(which is likely to grow exponentially) with two smaller sub graphs – one starting from initial vertex and other starting from goal vertex. The search terminates when two graphs intersect.

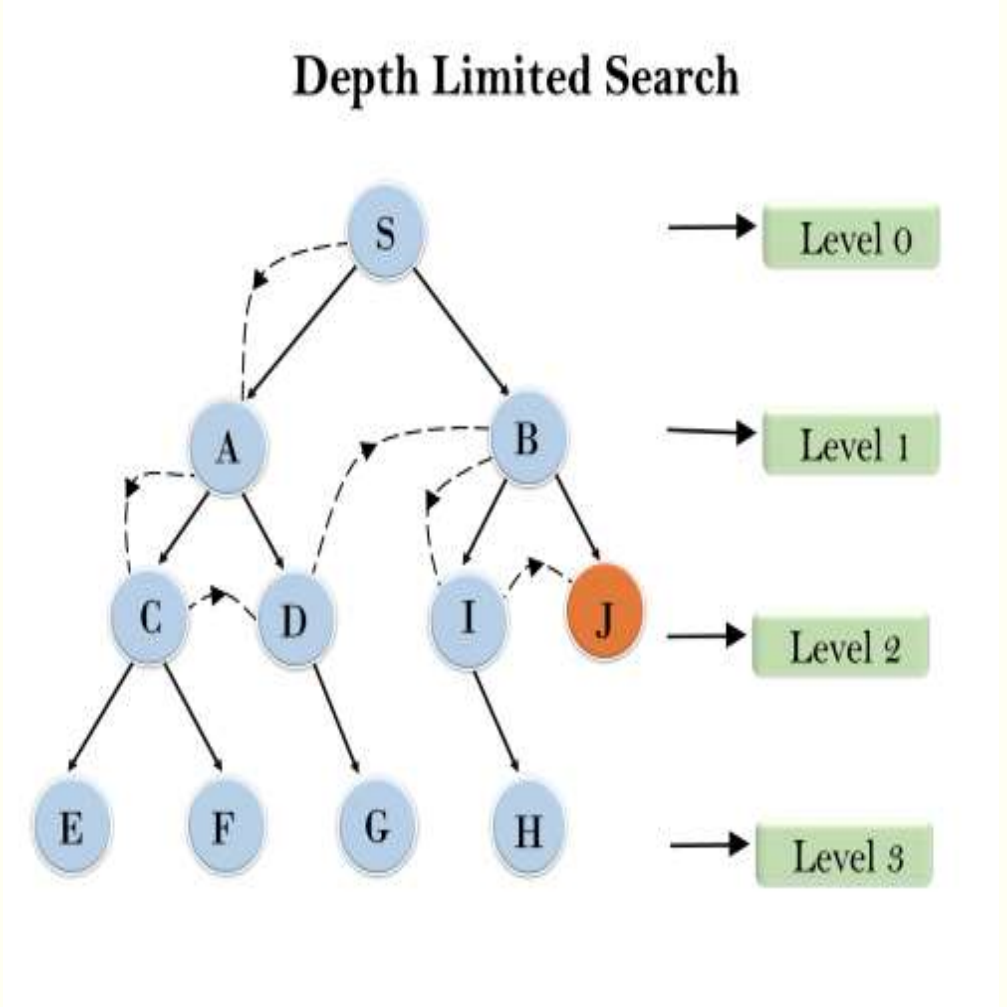
*Completeness : Bidirectional search is complete if BFS is used in both searches.
Optimality : It is optimal if BFS is used for search and paths have uniform cost.
Time and Space Complexity : Time and space complexity is O(bd/2)

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.



Depth-Limited Search Algorithm

Advantages:

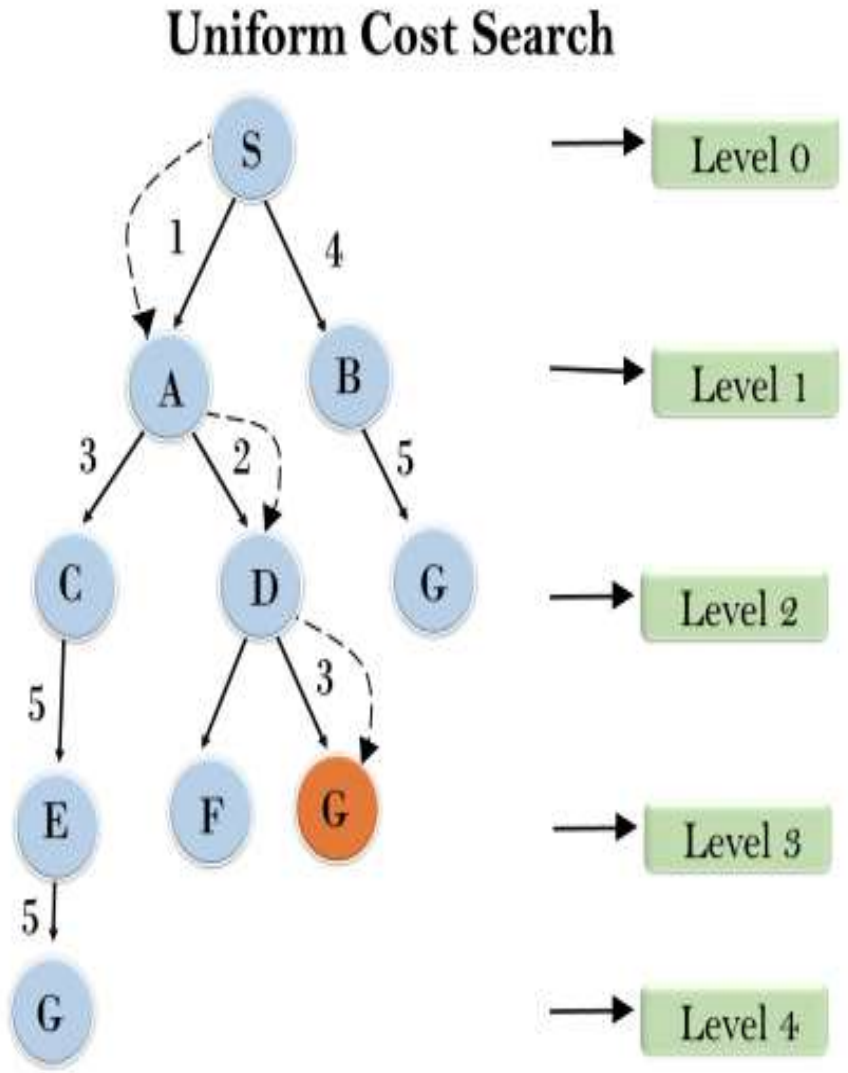
- Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.
- Completeness: DLS search algorithm is complete if the solution is above the depth-limit.
- Time Complexity: Time complexity of DLS algorithm is O(b^d).
- Space Complexity: Space complexity of DLS algorithm is O(b×d).

Uniform-cost Search Algorithm

- Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph.
- This algorithm comes into play when a different cost is available for each edge.
- The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost.
- Uniform-cost search expands nodes according to their path costs form the root node.
- It can be used to solve any graph/tree where the optimal cost is in demand.
- A uniform-cost search algorithm is implemented by the priority queue.
- It gives maximum priority to the lowest cumulative cost.
- Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.



Uniform-cost Search Algorithm

Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Time Complexity:

- Let C* is Cost of the optimal solution, and ε is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε.
- Hence, the worst-case time complexity of Uniform-cost search is O(b1 + [C*/ε])/.

Space Complexity:

- The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is O(b1 + [C*/ε]).

Optimal:

- Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

Heuristic Search Techniques in AI

- One of the core methods AI systems use to navigate problem-solving is through heuristic search techniques. These techniques are essential for tasks that involve finding the best path from a starting point to a goal state, such as in navigation systems, game playing, and optimization problems.

Understanding Heuristic Search

- Heuristics operates on the search space of a problem to find the best or closest-to-optimal solution via the use of systematic algorithms.
- A heuristic search method uses heuristic information to define a route that seems more plausible than the rest.
- Utilizing heuristic guiding, the algorithms determine the balance between exploration and exploitation, and thus they can successfully tackle demanding issues. Therefore, they enable an efficient solution finding process.

Heuristic Search Techniques in AI

Significance of Heuristic Search in AI

- The primary benefit of using heuristic search techniques in AI is their ability to handle large search spaces. Heuristics help to prioritize which paths are most likely to lead to a solution, significantly reducing the number of paths that must be explored. This not only speeds up the search process but also makes it feasible to solve problems that are otherwise too complex to handle with exact algorithms.

Components of Heuristic Search

Heuristic search algorithms typically comprise several essential components:

- State Space: This implies that the totality of all possible states or settings, which is considered to be the solution for the given problem.
- Initial State: The instance in the search tree of the highest level with no null values, serving as the initial state of the problem at hand.
- Goal Test: The exploration phase ensures whether the present state is a terminal or consenting state in which the problem is solved.

Heuristic Search Techniques in AI

Components of Heuristic Search

- Successor Function: This create a situation where individual states supplant the current state which represent the possible moves or solutions in the problem space.
- Heuristic Function: The function of a heuristic is to estimate the value or distance from a given state to the target state. It helps to focus the process on regions or states that has prospect of achieving the goal.

Types of Heuristic Search Techniques

1. **A* Search Algorithm**
2. **Greedy Best-First Search**
3. **Hill Climbing**
4. **Simulated Annealing**

A* Search Algorithm

A* Search Algorithm is perhaps the most well-known heuristic search algorithm. It uses a best-first search and finds the least-cost path from a given initial node to a target node. It has a heuristic function, often denoted as

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

where g(n) is the cost from the start node to n, and h(n) is a heuristic that estimates the cost of the cheapest path from n to the goal. A* is widely used in pathfinding and graph traversal.

What is the Greedy-Best-first search algorithm?

- The algorithm works by using a heuristic function to determine which path is the most promising.
- The heuristic function takes into account the cost of the current path and the estimated cost of the remaining paths.
- If the cost of the current path is lower than the estimated cost of the remaining paths, then the current path is chosen. This process is repeated until the goal is reached.

Greedy-Best-first search algorithm

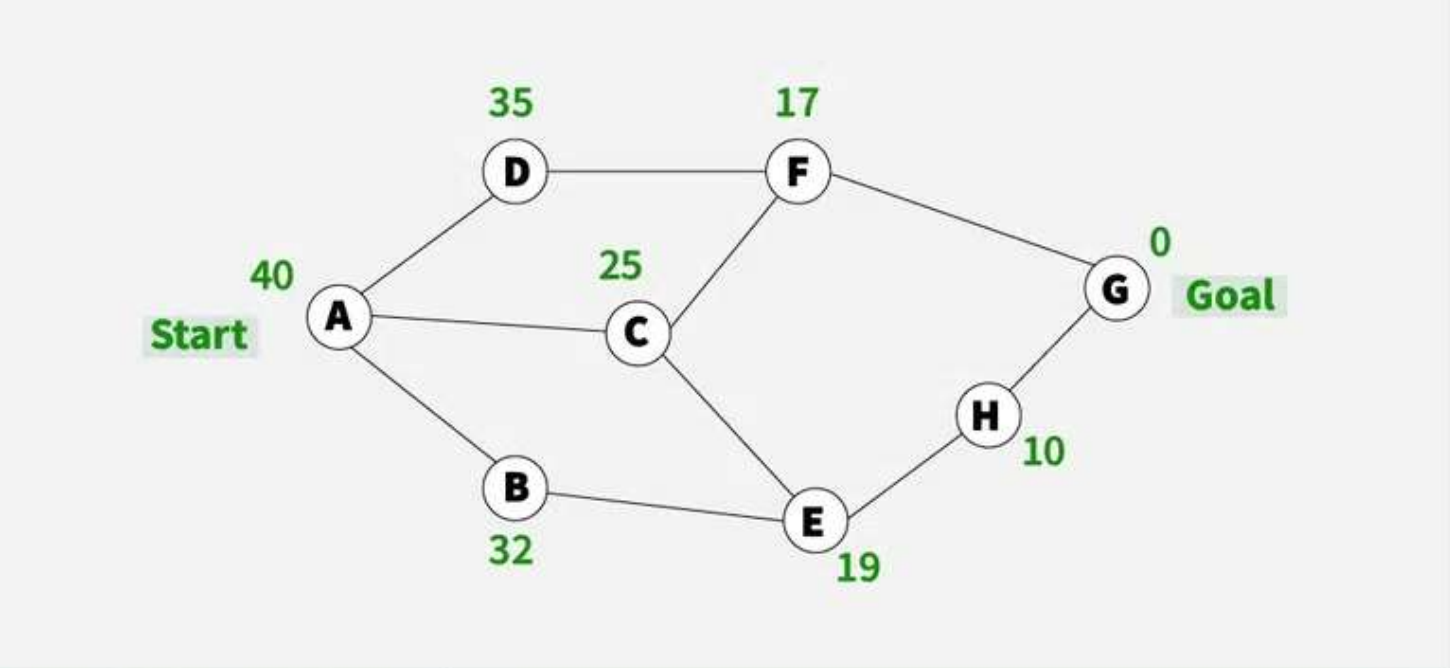
How Greedy Best-First Search Works?

- Greedy Best-First Search works by evaluating the cost of each possible path and then expanding the path with the lowest cost. This process is repeated until the goal is reached.
- The algorithm uses a heuristic function to determine which path is the most promising.
- The heuristic function takes into account the cost of the current path and the estimated cost of the remaining paths.
- If the cost of the current path is lower than the estimated cost of the remaining paths, then the current path is chosen. This process is repeated until the goal is reached.

Greedy-Best-first search algorithm

An example of the best-first search algorithm is below graph, suppose we have to find the path from A to G

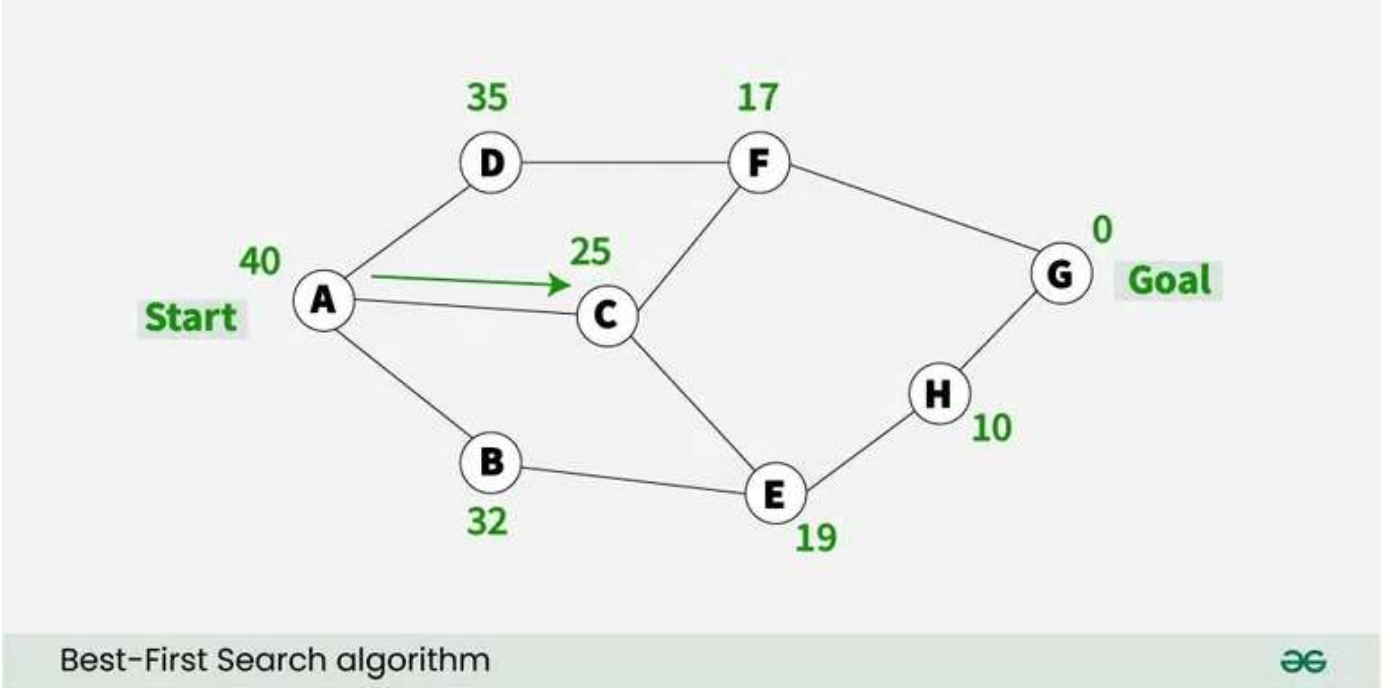
The values in green color represent the heuristic value of reaching the goal node G from current node



Greedy-Best-first search algorithm

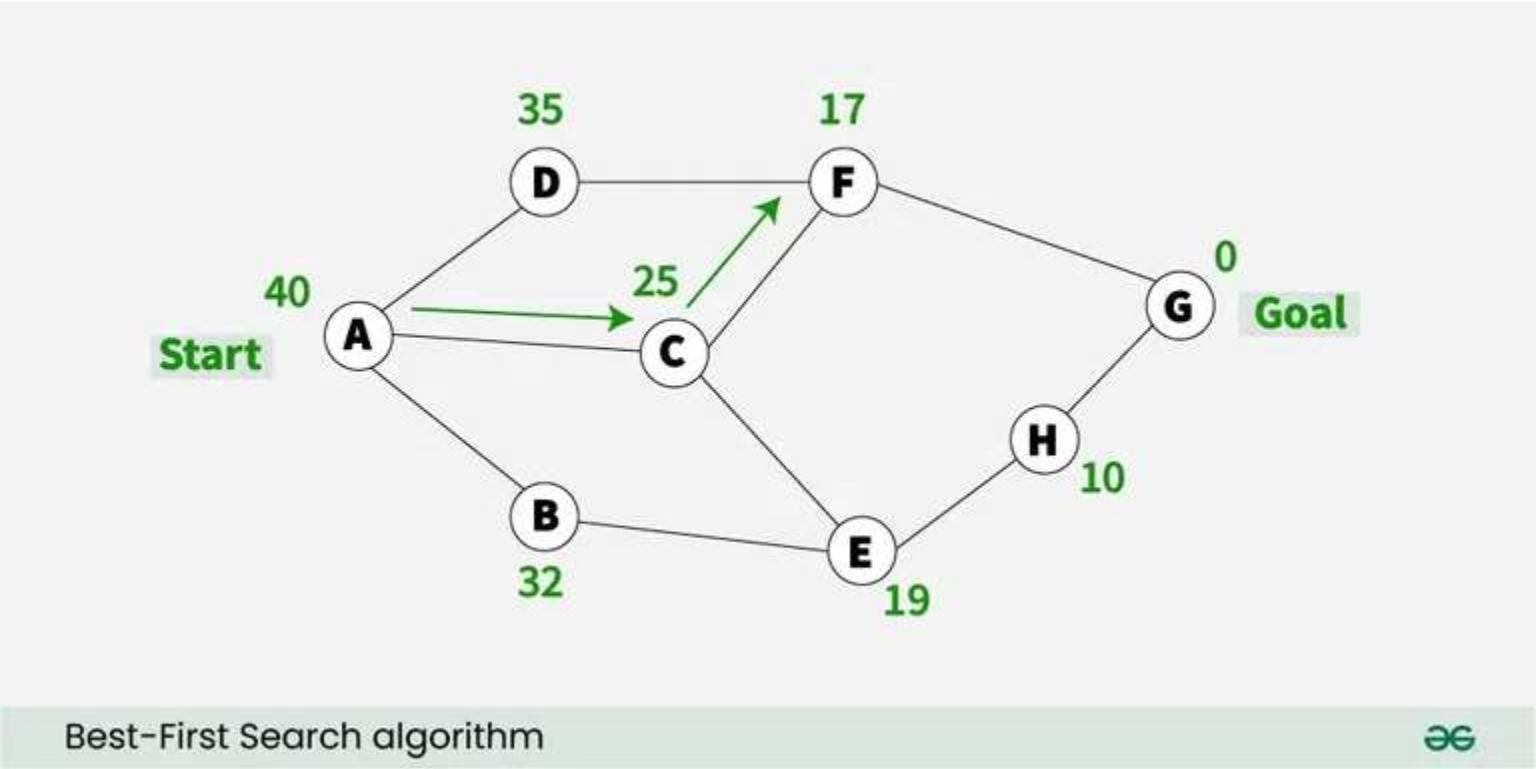
1) We are starting from A , so from A there are direct path to node B(with heuristics value of 32) , from A to C (with heuristics value of 25) and from A to D(with heuristics value of 35) .

2) So as per best first search algorithm choose the path with lowest heuristics value , currently C has lowest value among above node . So we will go from A to C.



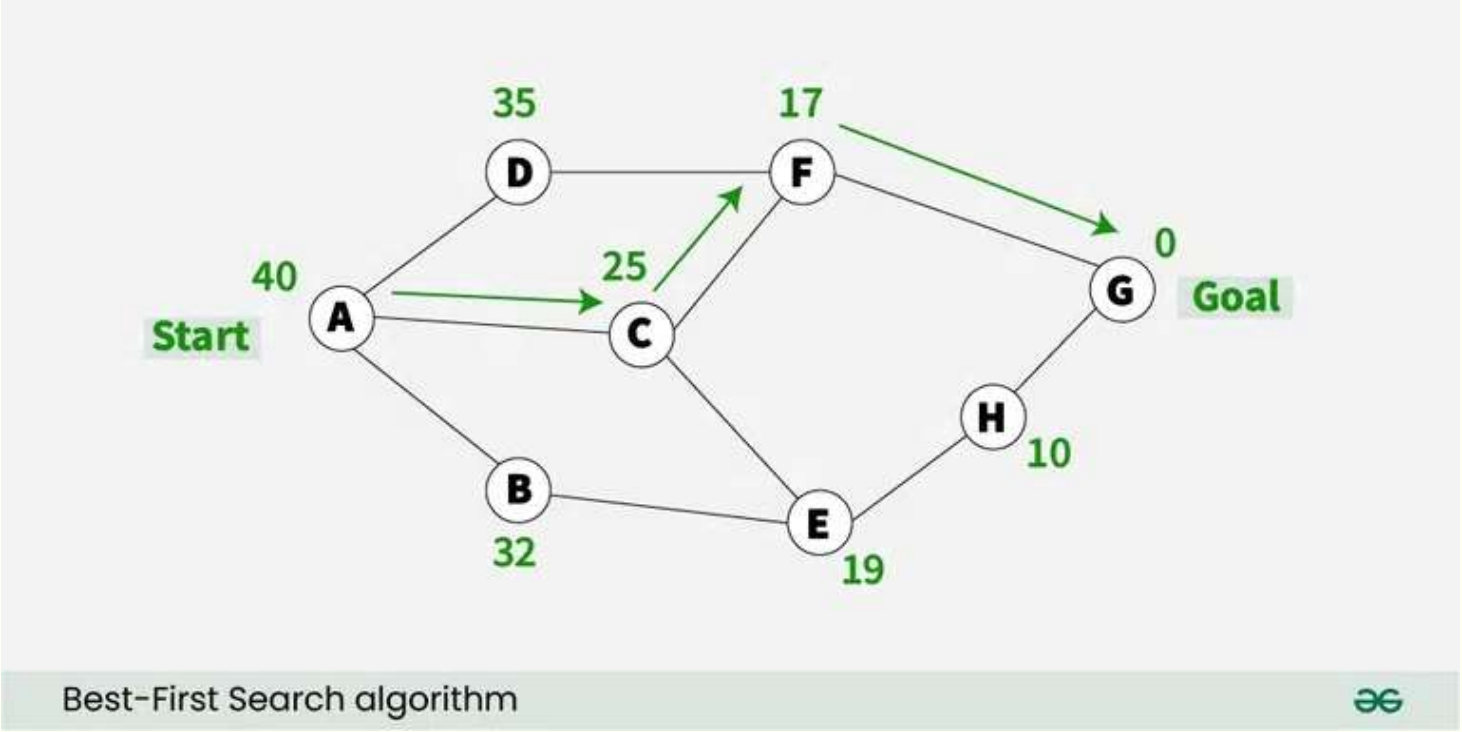
Greedy-Best-first search algorithm

3) Now from C we have direct paths as C to F(with heuristics value of 17) and C to E(with heuristics value of 19) , so we will go from C to F.



Greedy-Best-first search algorithm

4) Now from F we have direct path to go to the goal node G (with heuristics value of 0) , so we will go from F to G.



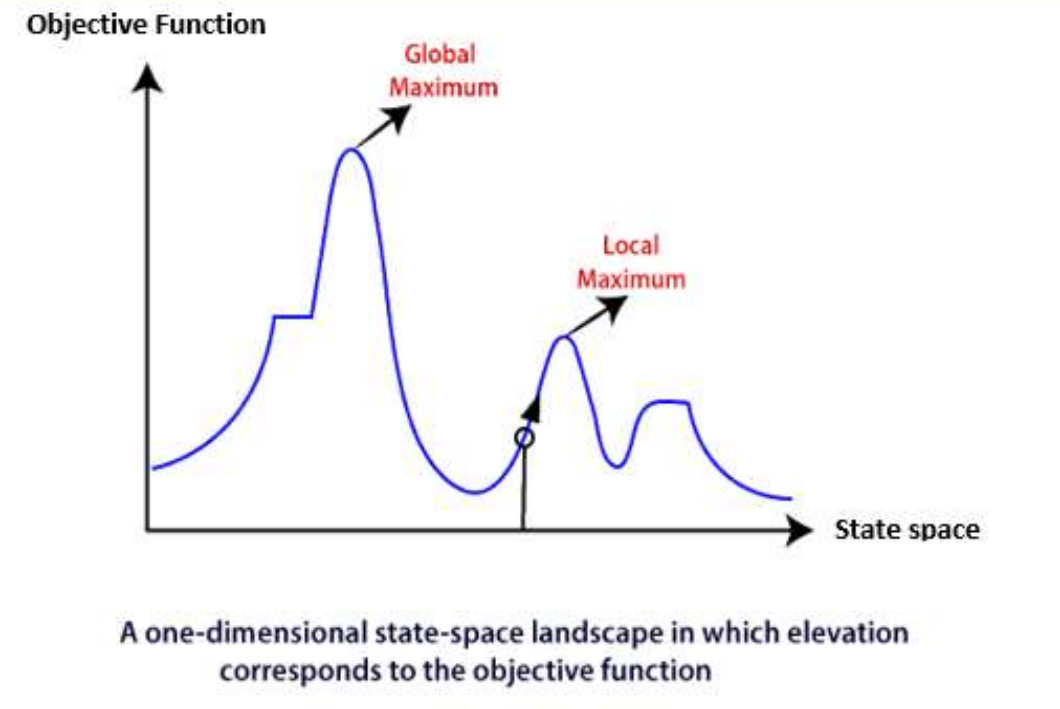
5) So now the goal node G has been reached and the path we will follow is A->C->F->G .

Local search Algorithm

- Local search algorithms are a category of optimization methods that focus on iteratively moving from one solution to a neighboring solution.
- They do this by evaluating solutions based on a heuristic function, which provides a measure of the solution's quality.
- These algorithms aim to find the best solution in the vicinity (the area near or surrounding a particular place) of the current state, rather than exploring the entire solution space.
- This local exploration strategy is what makes them particularly valuable.

Local search Algorithm

- In the context of local search algorithms, the "local" aspect refers to the limited scope of the search. These algorithms are designed to optimize within a constrained neighborhood of the current state, as opposed to global optimization methods that attempt to find the global optimum across the entire solution space.



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

Components of Local Search Algorithms

- **Initial State:** The initial state, also known as the starting point, is where the local search begins. Local search algorithms start with this initial state and iteratively explore neighboring solutions to improve upon it.
- **Neighbors:** Neighbors are solutions that are closely related to the current state. They are obtained by making small modifications to the current state, such as changing one element or moving to an adjacent node in a search space. Neighbors are essential because local search algorithms focus on refining the current solution by examining these nearby options.
- **Objective Function:** The objective function, also referred to as the evaluation function or heuristic function, plays a central role in local search algorithms. This function quantifies the quality or desirability of a solution. It assigns a numerical value to each solution, reflecting how close it is to the optimal solution. The objective function guides the search process by helping the algorithm select the most promising neighbors for exploration.

How These Components Work Together

- **Initialization:** The algorithm begins with an initial state, which is often randomly generated or provided based on problem requirements.
- **Evaluation:** The objective function is applied to the initial state, providing a numerical evaluation of its quality. This evaluation serves as a benchmark for comparison with neighboring solutions**.**
- **Exploration:** The algorithm explores the neighboring solutions of the current state. It generates these neighbors by making small, incremental changes to the current solution.
- **Selection:** After evaluating the neighboring solutions using the objective function, the algorithm selects the neighbor with the highest evaluation (indicating better quality) or other criteria that align with the optimization goal.
- **Update:** The selected neighbor becomes the new current state, and the process continues iteratively. The algorithm repeats the evaluation, exploration, and selection steps until a termination condition is met. Termination conditions may include reaching a predefined number of iterations, finding a satisfactory solution, or running out of computational resources.

Step-by-Step Workflow of a Local Search Algorithm

Local search algorithms follow a systematic workflow to search for optimal or near-optimal solutions within a problem space. Let's break down this process step by step:

- **Start with an Initial State:**

The local search algorithm in AI begins with an initial state. This state represents a possible solution to the problem and serves as the starting point for the search.

- **Evaluate Neighboring States:**

The algorithm evaluates the quality of the current state by applying the objective function (heuristic evaluation function). The result of this evaluation provides a measure of how close the current state is to the optimal solution.

Step-by-Step Workflow of a Local Search Algorithm

- **Move to the Neighbor with the Best Evaluation:**

The algorithm explores neighboring states by making incremental changes to the current state. These changes are guided by the problem's constraints and the objective function.

The neighboring states are evaluated using the objective function, and the one with the best evaluation (indicating higher quality or closeness to the optimal solution) is selected as the next current state.

This process of evaluation, exploration, and selection is repeated iteratively. The algorithm keeps moving to the neighbor with the best evaluation in each iteration.

- **Repeat Until a Solution is Found or Termination Condition is Met**:

The local search algorithm continues the cycle of evaluating neighboring states, selecting the best one, and updating the current state until it either finds a satisfactory solution or meets a termination condition.

Termination conditions can vary and may include reaching a predefined number of iterations, finding a solution that meets certain criteria, or exhausting available computational resources.

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.
- 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.

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.

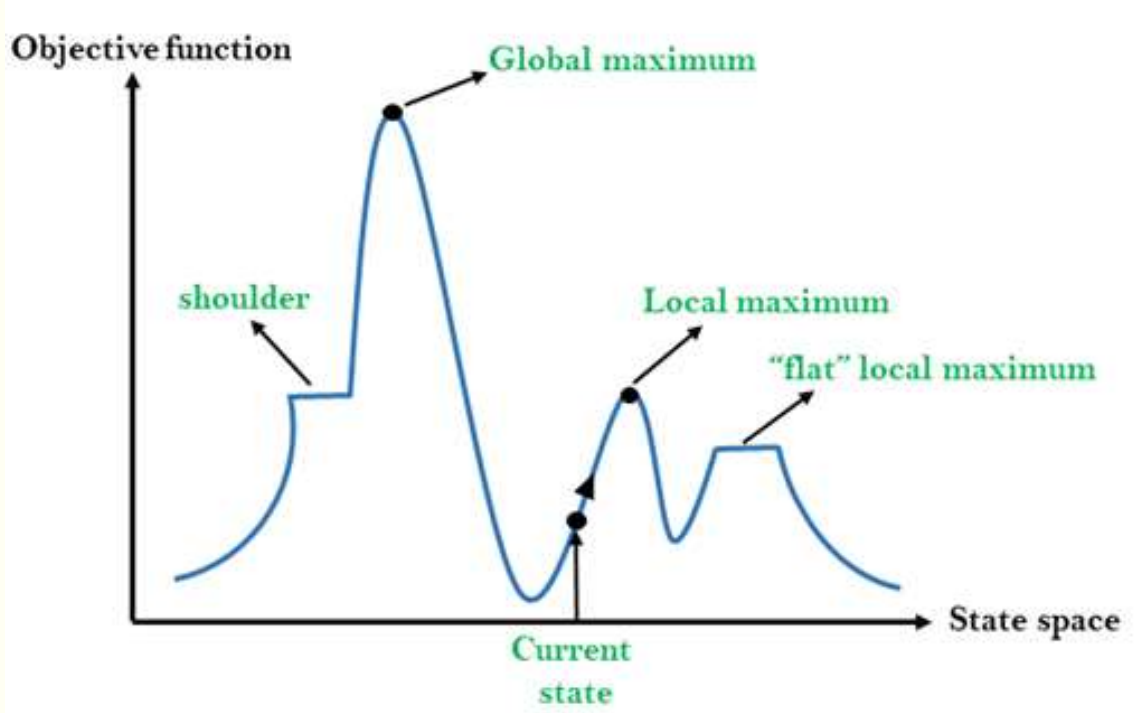
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.

- **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.

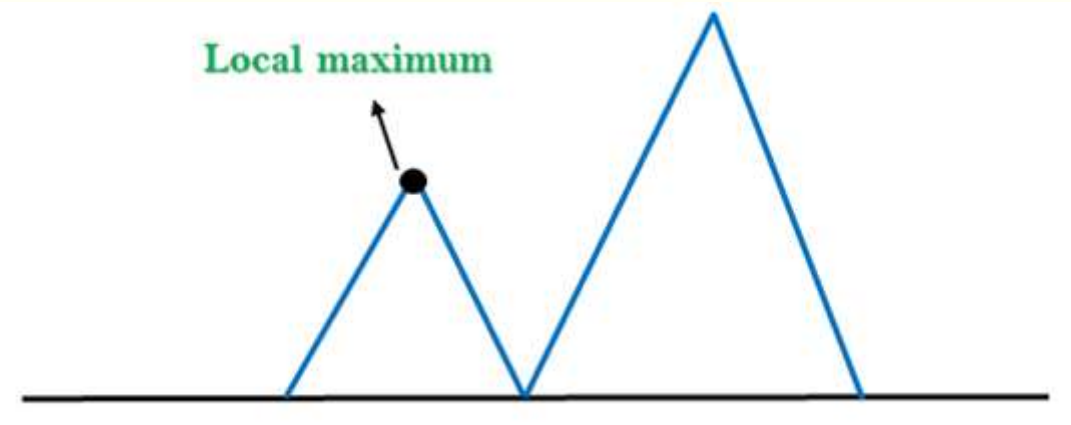
Current state: It is a state in a landscape diagram where an agent is currently present.

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.

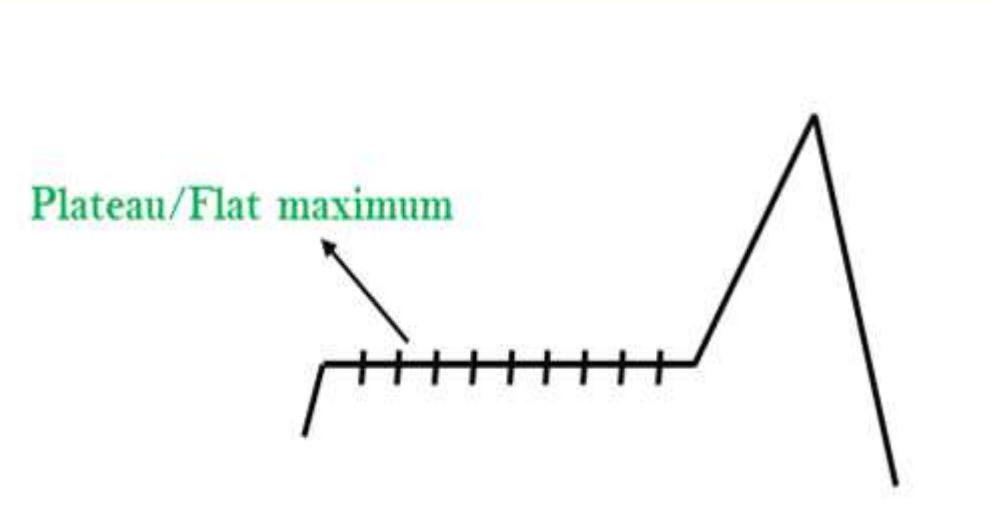
Problems in Hill Climbing Algorithm:

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



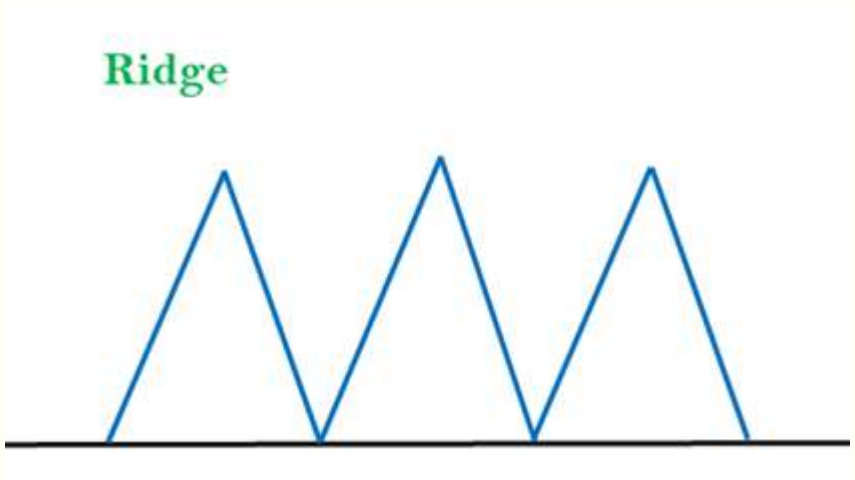
Problems in Hill Climbing Algorithm:

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.



Problems in Hill Climbing Algorithm:

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.



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.

THANK YOU !!

Prof. U. M Rane

umrane@kkwagh.edu.in

K.K.Wagh Institute of Engineering Education & Research, Nashik