PRINCIPLES OF ARTIFICIAL INTELLIGENCE

Unit 2

Search Algorithms

Search Algorithms in Artificial Intelligence

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.

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.

Toy problems

-Vacuum cleaner agent(covered in unit I)

-8 puzzle

-n-queens

-Crypt arithmetic (chalk and talk)

-Missionaries cannibals

Real world problems

-Route finding

-Traveling salesperson

-VLSI layout

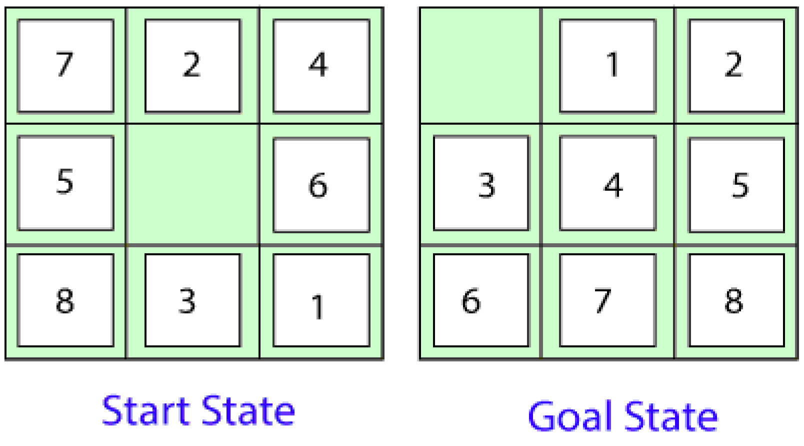
-Robot navigation

-Assembly sequencing

Some Toy Problems

8 Puzzle Problem: Here, we have a 3x3 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:

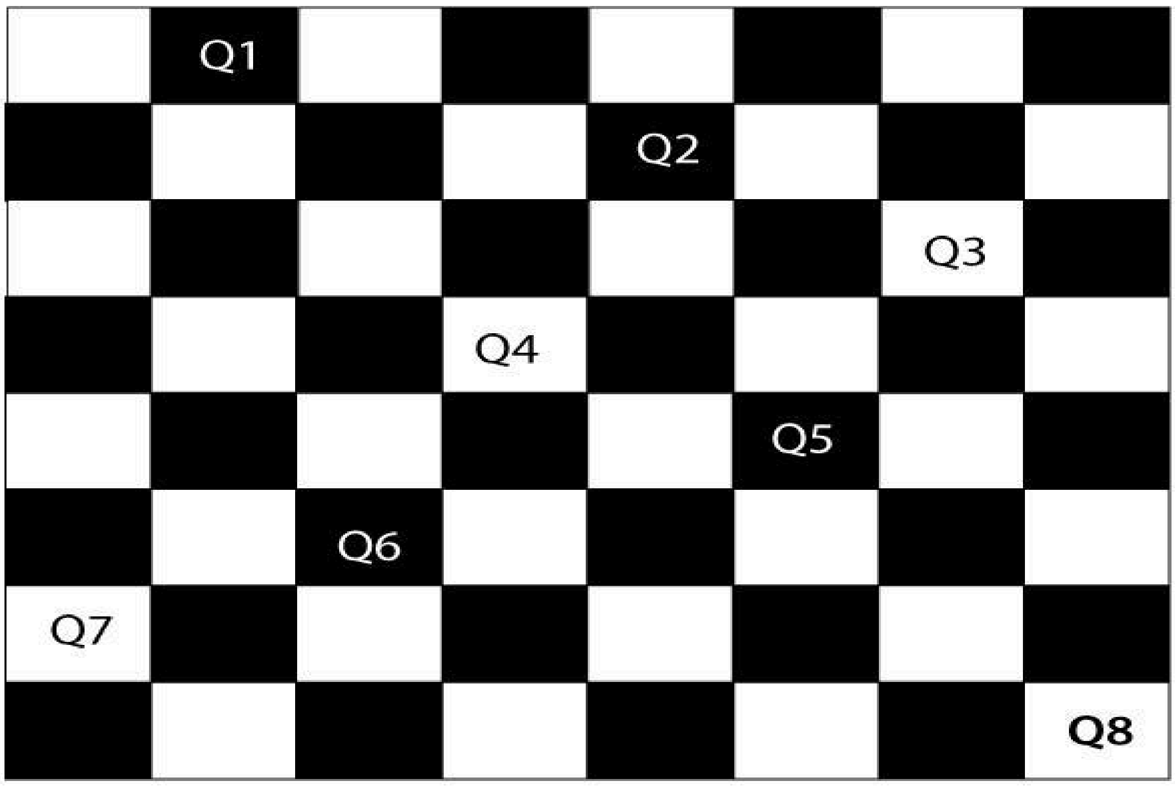
- **States:** It describes the location of each numbered tiles and the blank tile.
- **Initial State:** We can start from any state as the initial state.
- **Actions:** Here, actions of the blank space is defined, i.e., either **left, right, up or down**
- **Transition Model:** It returns the resulting 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:

- **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.

- **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.

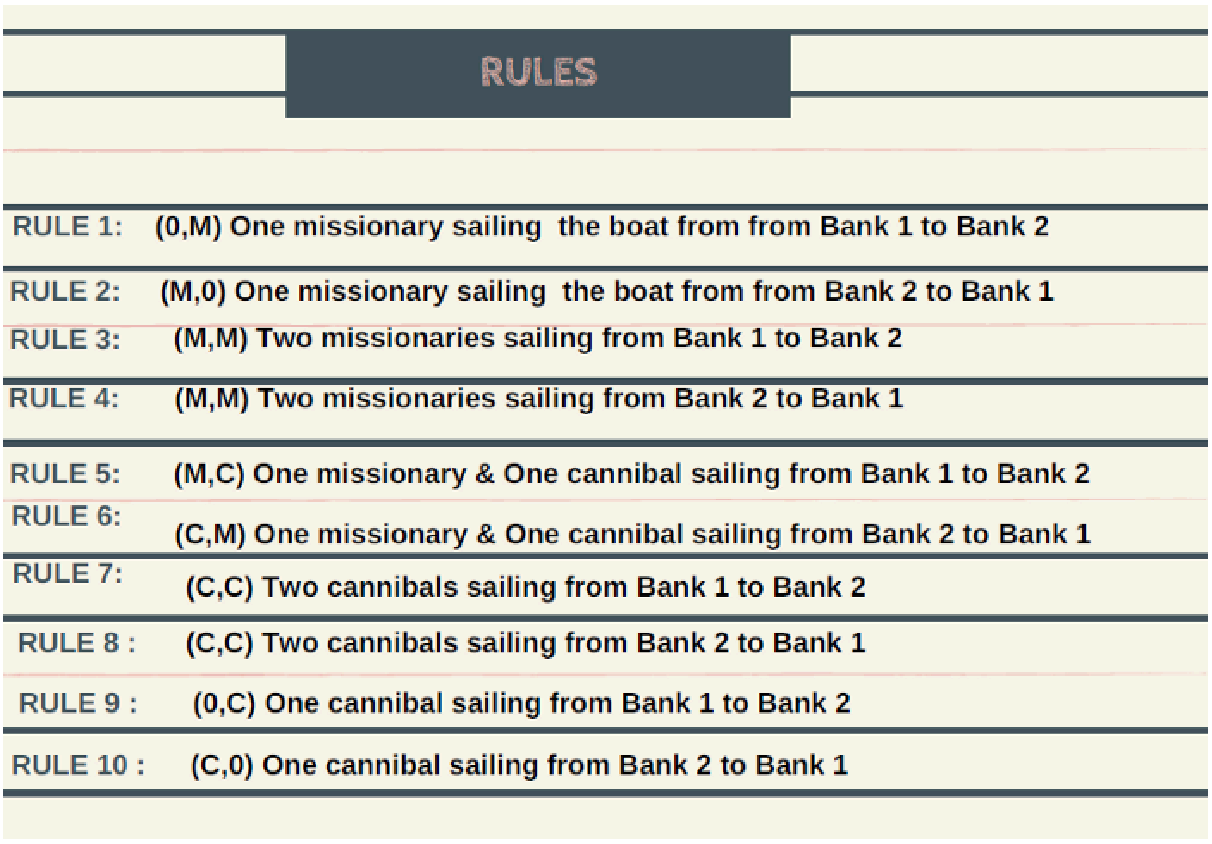
This formulation is better than the incremental formulation as it reduces the state space from **1.8×10^{14}** to **2057**, and it is easy to find the solutions.

The Missionaries and Cannibals Problem.

On one bank of a river are three missionaries and three cannibals find themselves and need to cross the river. The only boat available holds only two at a time. And If the cannibals ever outnumber the missionaries on either of the river's banks, the missionaries will get eaten.

How can everyone get across the river without the missionaries risking being eaten?

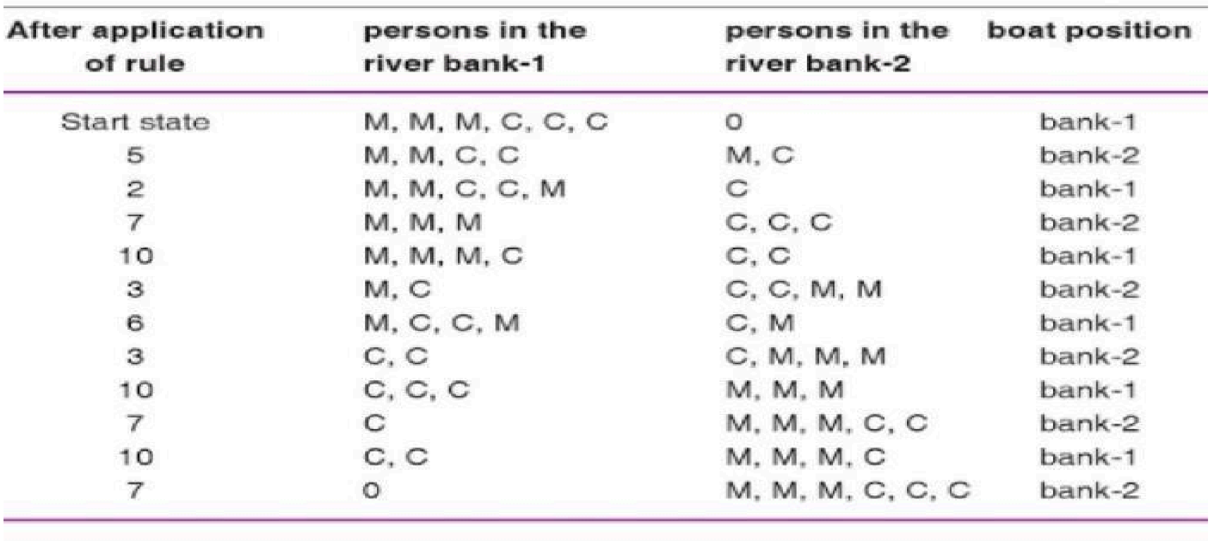
Three missionaries and three cannibals present at Bank 1 of river.



RULES

RULE 1:	(0,M) One missionary sailing the boat from from Bank 1 to Bank 2
RULE 2:	(M,0) One missionary sailing the boat from from Bank 2 to Bank 1
RULE 3:	(M,M) Two missionaries sailing from Bank 1 to Bank 2
RULE 4:	(M,M) Two missionaries sailing from Bank 2 to Bank 1
RULE 5:	(M,C) One missionary & One cannibal sailing from Bank 1 to Bank 2
RULE 6:	(C,M) One missionary & One cannibal sailing from Bank 2 to Bank 1
RULE 7:	(C,C) Two cannibals sailing from Bank 1 to Bank 2
RULE 8 :	(C,C) Two cannibals sailing from Bank 2 to Bank 1
RULE 9 :	(0,C) One cannibal sailing from Bank 1 to Bank 2
RULE 10 :	(C,0) One cannibal sailing from Bank 2 to Bank 1

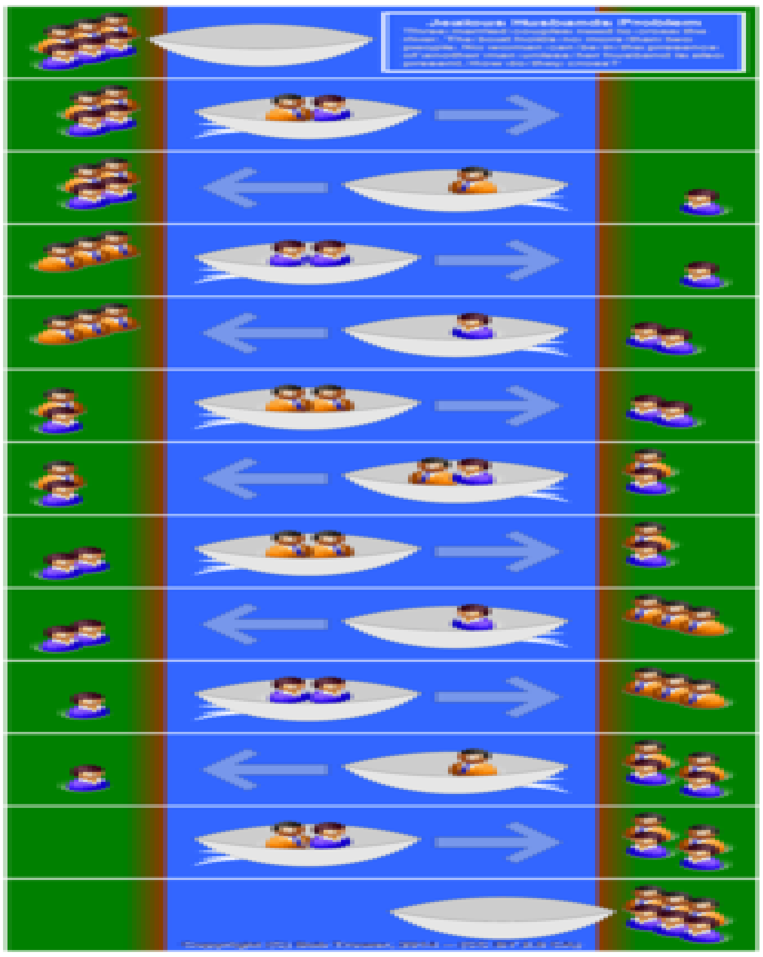
(Missionary is denoted by M and Cannibal is denoted by C)



After application of rule	persons in the river bank-1	persons in the river bank-2	boat position
Start state	M, M, M, C, C, C	0	bank-1
5	M, M, C, C	M, C	bank-2
2	M, M, C, C, M	C	bank-1
7	M, M, M	C, C, C	bank-2
10	M, M, M, C	C, C	bank-1
3	M, C	C, C, M, M	bank-2
6	M, C, C, M	C, M	bank-1
3	C, C	C, M, M, M	bank-2
10	C, C, C	M, M, M	bank-1
7	C	M, M, M, C, C	bank-2
10	C, C	M, M, M, C	bank-1
7	0	M, M, M, C, C, C	bank-2

(After applying the rules)

Lets play the game :



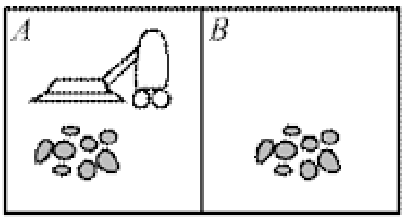
Vacuum cleaner

Vaccum cleaner problem

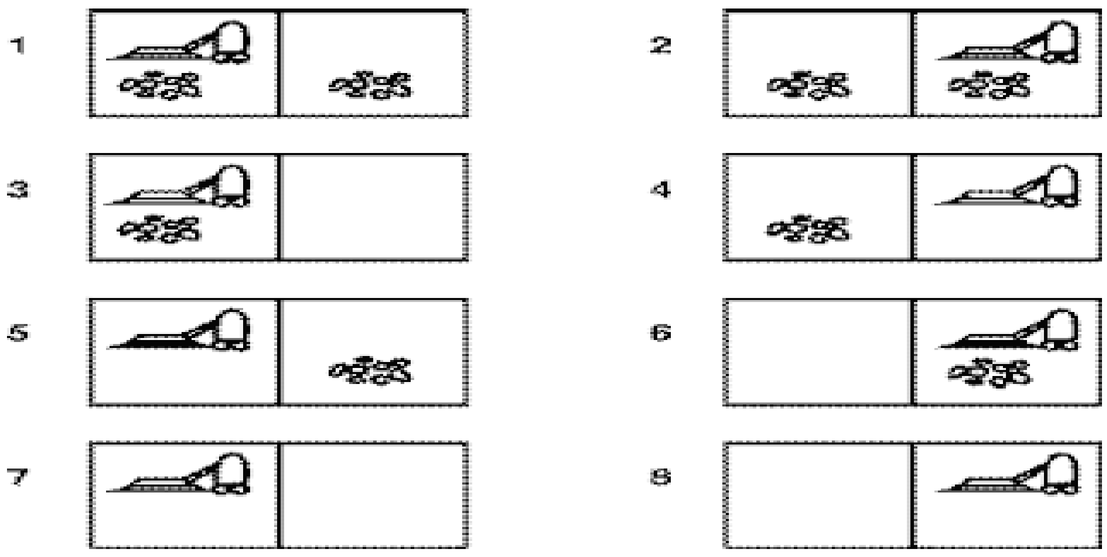
- Vacuum cleaner problem is a well-known search problem for an agent which works on Artificial Intelligence. In this problem, our vacuum cleaner is our agent. It is a goal based agent, and the goal of this agent, which is the vacuum cleaner, is to clean up the whole area. So, in the classical vacuum cleaner problem, we have two rooms and one vacuum cleaner. There is dirt in both the rooms and it is to be cleaned. The vacuum cleaner is present in any one of these rooms. So, we have to reach a state in which both the rooms are clean and are dust free.

- So, there are eight possible states possible in our vacuum cleaner problem. These can be well illustrated with the help of the following diagrams:

Consider a Vacuum cleaner world



Imagine that our intelligent agent is a robot vacuum cleaner.
Let's suppose that the world has just two rooms. The robot can be in either room and there can be dirt in zero, one, or two rooms.



- states 1 and 2 are our initial states
- state 7 and state 8 are our final states (goal states)
- both the rooms are full of dirt and the vacuum cleaner can reside in any room. And to reach the final goal state, both the rooms should be clean and the vacuum cleaner again can reside in any of the two rooms.
- 1.dirty,dirty

2.dirty,clean

3.clean,dirty

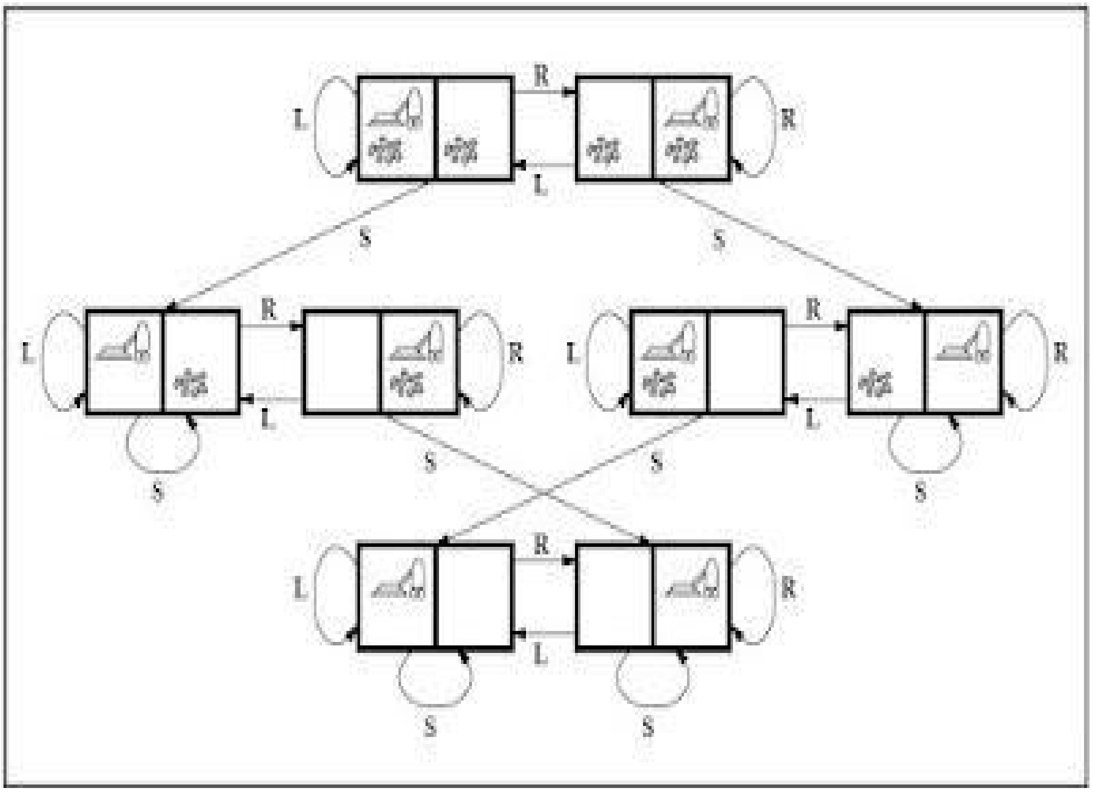
4.clean,clean

The vacuum cleaner can perform the following functions: move left, move right, move forward, move backward and to suck dust

Goal formulation: intuitively, we want all the dirt cleaned up. Formally, the goal is
{ state 7, state 8 }.

Problem formulation(Actions): Left,Right,Suck,NoOp

State Space Graph:



In the real world Problems

- The real world is absurdly complex.

  - Real state space must be abstracted for problem solving.

-An abstract state is equivalent to a set of real states.

- Abstract operator is equivalent to a complex combination of real actions.

-Robot operator: Move down hall

-In practice, this might involve a complex set of sensor and motor activities.

- An abstract solution is equivalent to a set of real paths that are solutions in the real world.

Example: Route finding problem

States: Each state obviously includes a location and the current time. Further more, because the cost of an action may depend on previous segments, their fare bases, and their state as domestic or international, the state must record extra information about these “historical” aspects.

Initial state: This is specified by user query.

Actions: Taking any flight from the current location, in any seat class, leaving after the current time.

Transition model: The state resulting from taking a flight will have the flight’s destination as the current location and the flight’s arrival time as the current time.

Goal test: are we at the final destination specified by the user?

Path cost: This depends on monetary cost, waiting time ,customers and immigration procedures seat quality, type of airplane etc..

Example: TSM in Romania

- On holiday in Romania; currently in Arad.

- **Formulate goal:**

- be in Bucharest

- **Formulate problem:**

- -states: various cities

-actions: drive between cities

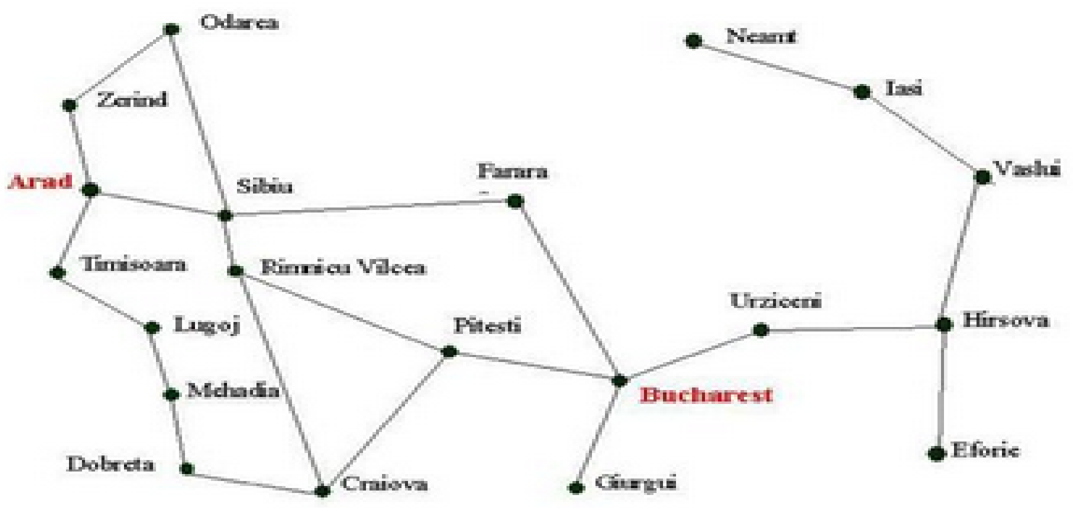
- **Find solution:**

-Sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

-**Goal Test** > Are we in Bucharest.

-**Cost Function** > Sum of road lengths to the destination

TSM IN ROMANIA contd…



Example: A VLSI LAYOUT

- A VLSI problem requires positioning millions of components and connections on a chip to minimize area, minimize circuit delays and manufacturing yield.

- The layout problems comes after the logical design phase and is usually split into two parts:

1)cell layout

2)channel routing

In cell layout, the primitive components of the circuits are grouped into cells, each of which performs some recognized function.

- Each cell has fixed footprint(size and shape) and requires a certain number of connections to each of the other cells.
- The aim is to place the cells on the chip so that they do not overlap and so that there is room for each wire through the gaps between the cells.

Channel routing finds a specific route for each wire through the gaps between the cells.

Example: Robot Navigation

- A robot can move in a continuous space with an infinite set of possible actions and states.
- For a circular robot moving on a flat surface, the space is essentially two-dimensional.
- When the robot has arms and legs or wheels that must be controlled, the search space becomes many-dimensional.

Advanced technique are required just to make the search space finite.

Example: Assembly sequencing

- In assembly problems, the aim is to find an order in which to assemble the parts of some object.
- If the wrong order chosen, there will be no way to add some part later in the sequence without undoing some of the work already done.
- Checking a step in the sequence for feasibility is a difficult geometrical search problem closely related to robot navigation.

Thus, the generation of legal actions is the expensive part of assembly sequencing.

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.

a. **Start State:** It is a state from where agent begins **the search**.

b. **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.

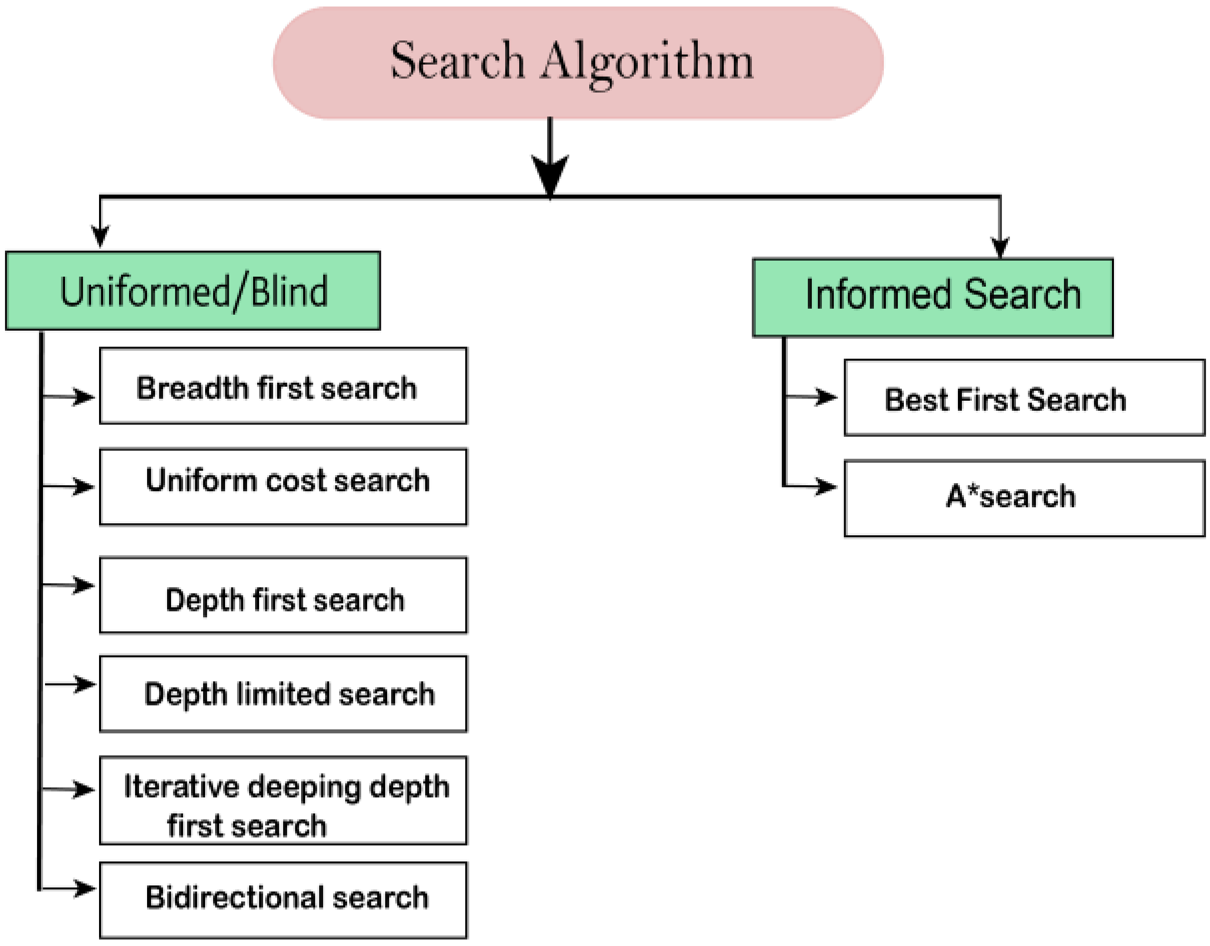
Optimality: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

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.

Types of search algorithms

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.



Uninformed/Blind Search:

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search.It examines each node of the tree until it achieves the goal node.

It can be divided into five main types:

- Breadth-first search
- Uniform cost search
- Depth-first search
- Iterative deepening depth-first search
- Bidirectional 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.

An example of informed search algorithms is a traveling salesman problem.

1. Greedy Search
2. A* Search

Chapter 2

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**
5. **Uniform cost search**
6. **Bidirectional Search**

1. 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.

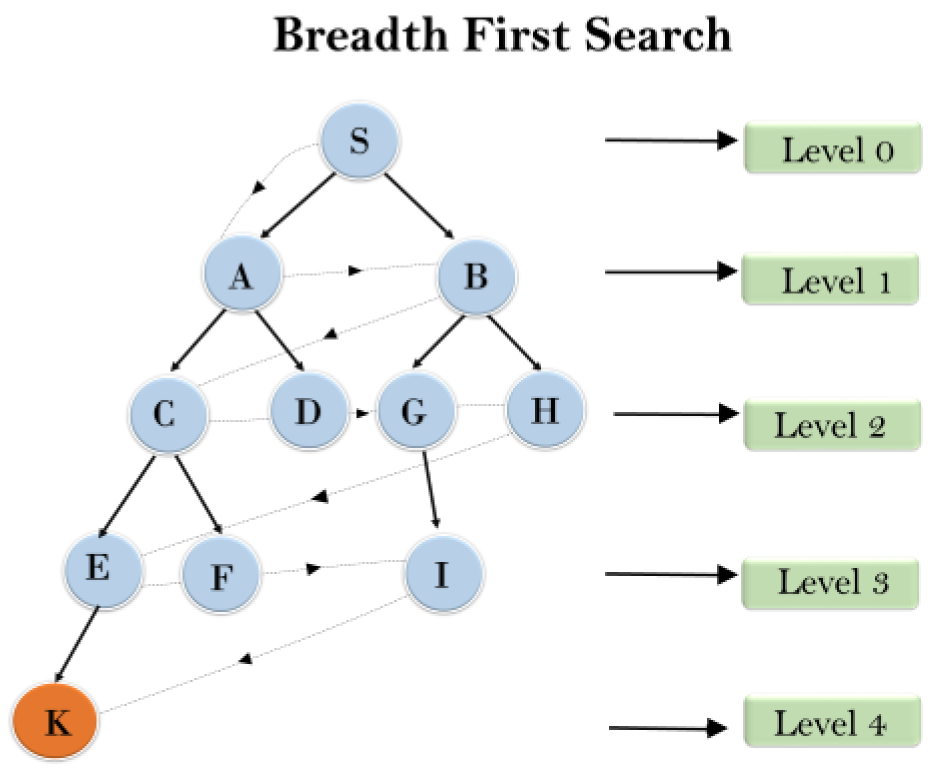
Disadvantages:

- 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:

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



Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

$T(b) = 1+b^2+b^3+.......+ b^d= O(b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

2. Depth-first Search

- Depth-first search isa 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.

Note: Backtracking is an algorithm technique for finding all possible solutions using recursion.

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:

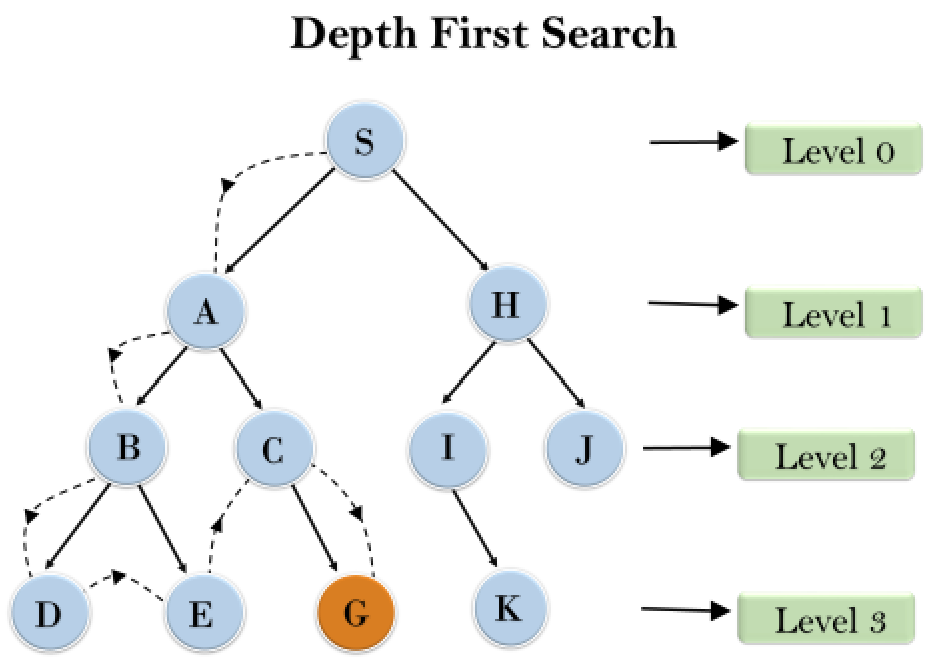
- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

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.



Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$T(n)= 1+ n^2+ n^3 +.........+ n^m=O(n^m)$

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

3. 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.

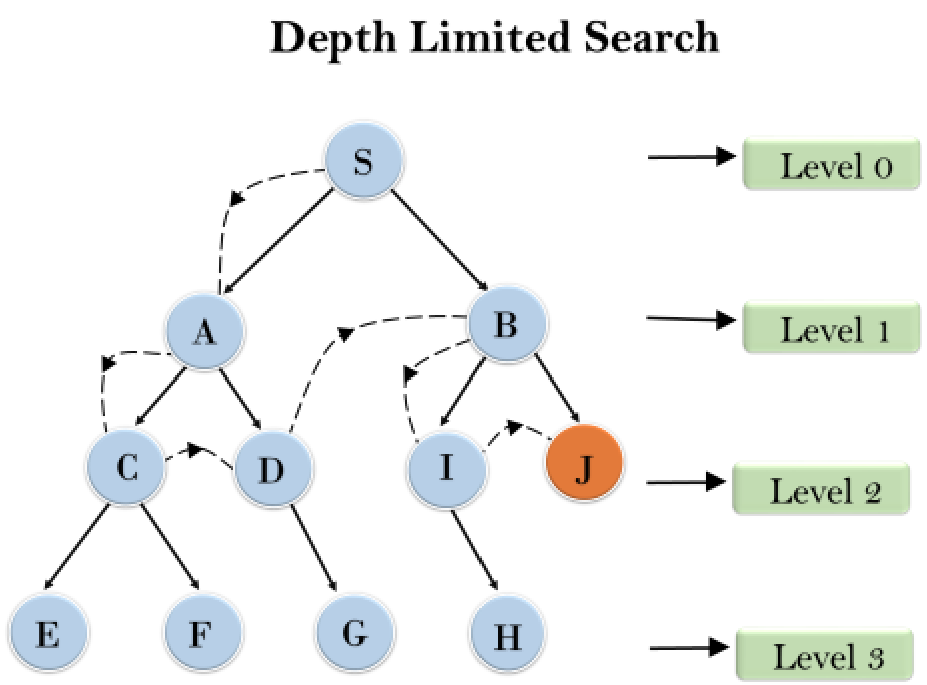
Advantages:

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:



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is order-**$O(b^{\ell})$**.

Space Complexity: Space complexity of DLS algorithm is O(**b×ℓ**).

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d.

4. 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.

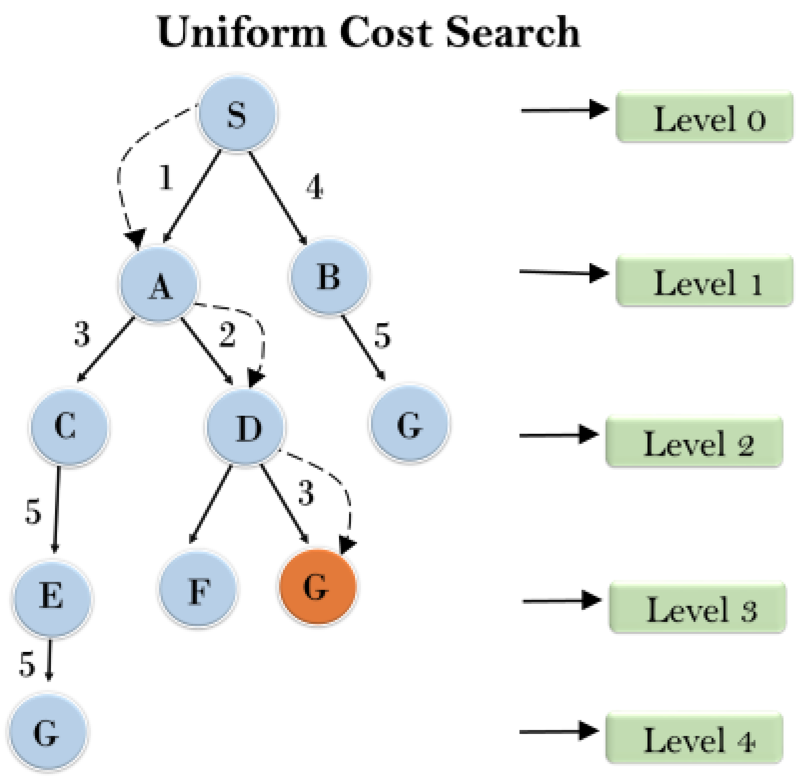
Advantages:

- 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.

Example:



Completeness:

Uniform-cost search is complete, such as if there is a solution, UCS will find it.

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**$O(b^{1 + [C^*/\varepsilon]})$**/.

Space Complexity:

The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is **$O(b^{1 + [C^*/\varepsilon]})$**.

Optimal:

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

5. Iterative deepening depth-first Search:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Advantages:

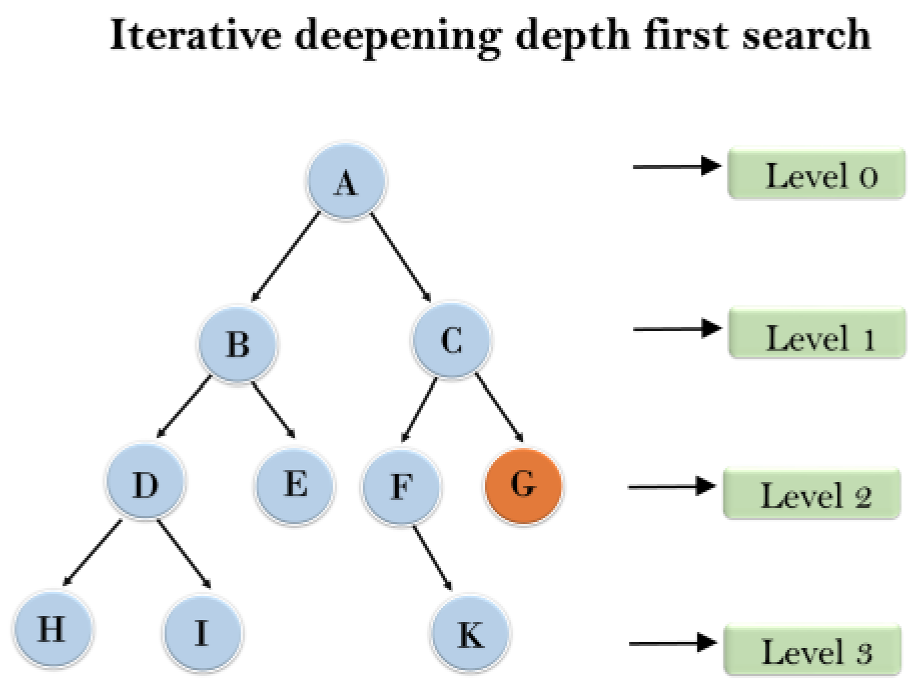
- Itcombines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:



1'st Iteration-----> A
2'nd Iteration----> A, B, C
3'rd Iteration------>A, B, D, E, C, F, G
4'th Iteration------>A, B, D, H, I, E, C, F, K, G
In the fourth iteration, the algorithm will find the goal node.

Completeness:

This algorithm is complete is ifthe branching factor is finite.

Time Complexity:

Let's suppose b is the branching factor and depth is d then the worst-case time complexity is **O(b^d)**.

Space Complexity:

The space complexity of IDDFS will be **O(bd)**.

Optimal:

IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

6. Bidirectional Search Algorithm:

Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

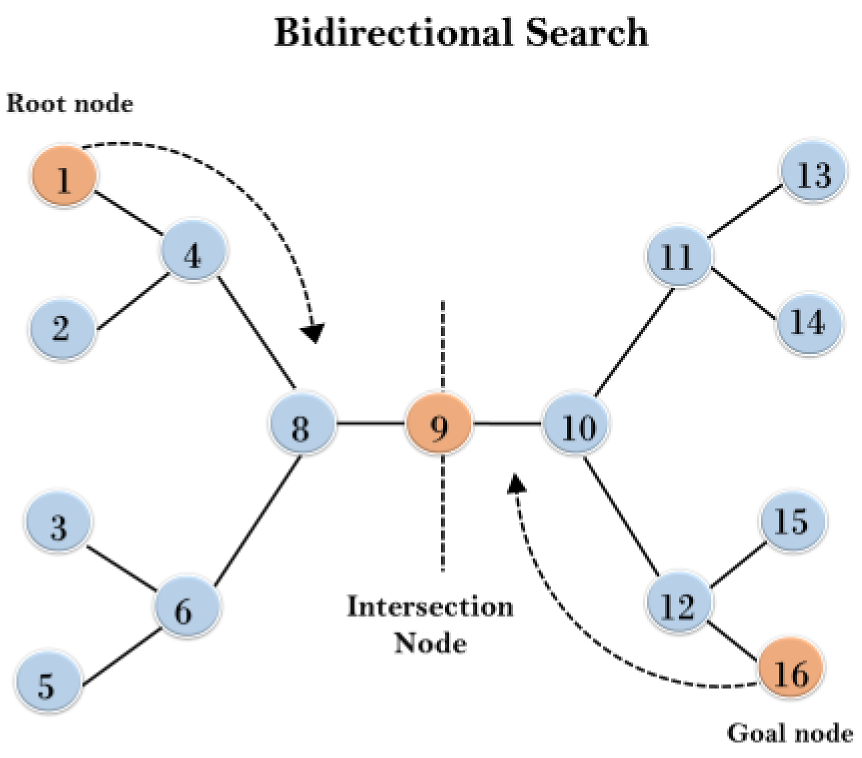
Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- **In bidirectional search, one should know the goal state in advance.**

Example:

In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

The algorithm terminates at node 9 where two searches meet.



Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS is **$O(b^d)$**.

Space Complexity: Space complexity of bidirectional search is **$O(b^d)$**.

Optimal: Bidirectional search is Optimal.

Chapter 3

Informed Search Algorithms

So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But 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.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic 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 two main algorithms which are given below:

- **Best First Search Algorithm(Greedy search)**
- **A* Search Algorithm**

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.

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:

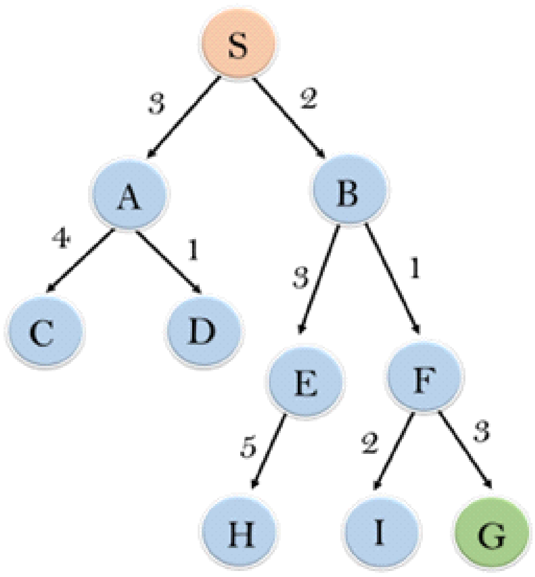
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

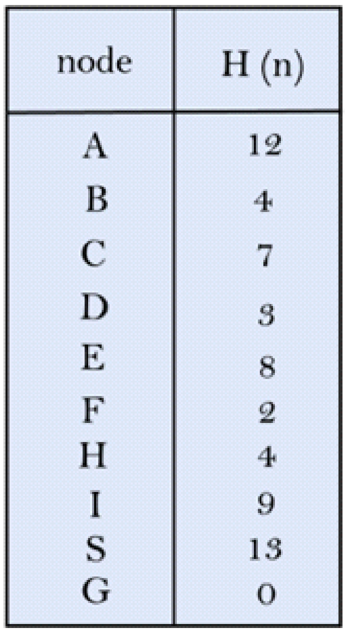
Disadvantages:

- 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:

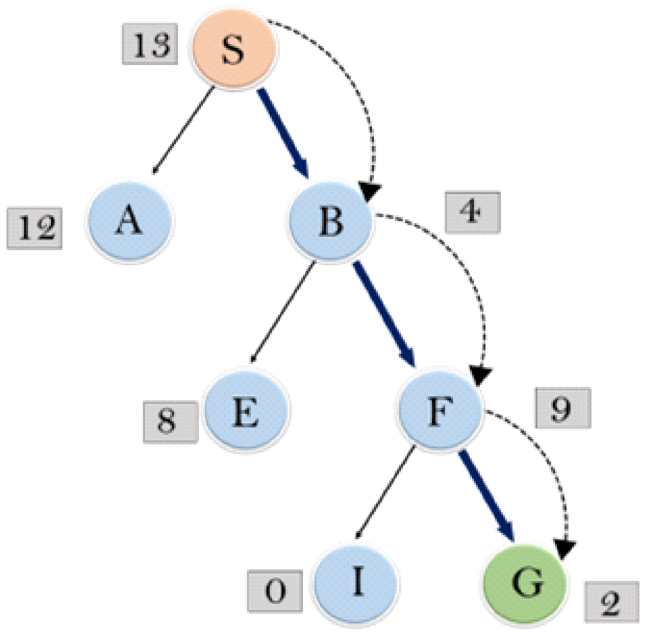
Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.





node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

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

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

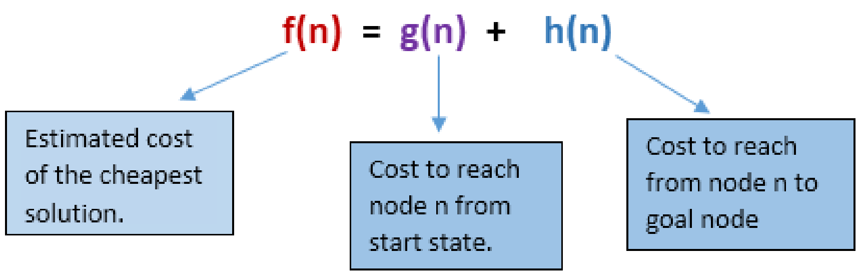
Iteration state space is finite.

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**.


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

At each point in the search space, only those node is expanded which have the lowest value of f(n), and the algorithm terminates when the goal node is found.

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 (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:

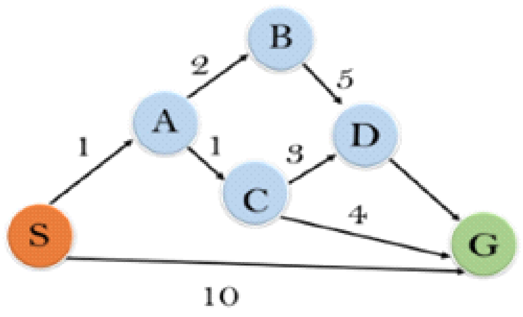
- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

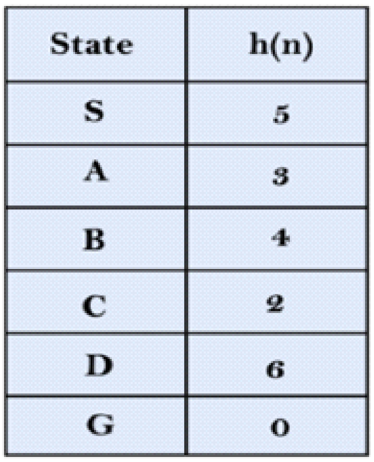
Disadvantages:

- 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:

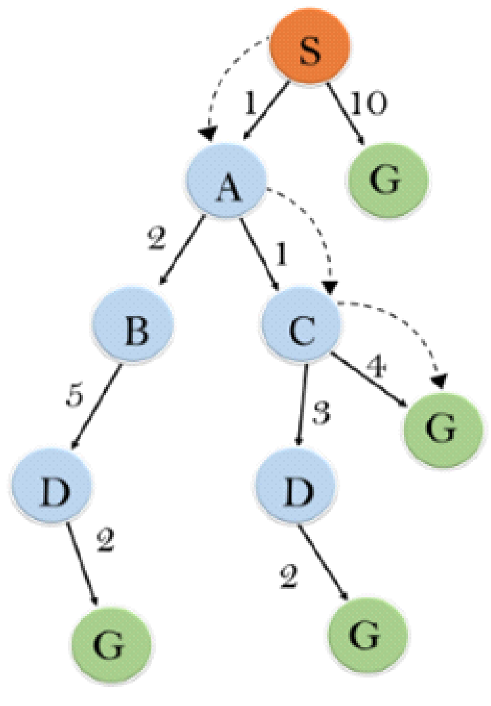
In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.
Here we will use OPEN and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

Means-Ends Analysis in Artificial Intelligence

We have studied the strategies which can reason either in forward or backward, but a mixture of the two directions is appropriate for solving a complex and large problem. Such a mixed strategy, make it possible that first to solve the major part of a problem and then go back and solve the small problems arise during combining the big parts of the problem. Such a technique is called **Means-Ends Analysis**.

- Means-Ends Analysis is problem-solving techniques used in Artificial intelligence for limiting search in AI programs.
- It is a mixture of Backward and forward search technique.
- The MEA technique was first introduced in 1961 by Allen Newell, and Herbert A. Simon in their problem-solving computer program, which was named as General Problem Solver (GPS).
- The MEA analysis process centered on the evaluation of the difference between the current state and goal state.

How means-ends analysis Works:

The means-ends analysis process can be applied recursively for a problem. It is a strategy to control search in problem-solving. Following are the main Steps which describes the working of MEA technique for solving a problem.

a. First, evaluate the difference between Initial State and final State.

b. Select the various operators which can be applied for each difference.

c. Apply the operator at each difference, which reduces the difference between the current state and goal state.

Operator Subgoaling

In the MEA process, we detect the differences between the current state and goal state. Once these differences occur, then we can apply an operator to reduce the differences. But sometimes it is possible that an operator cannot be applied to the current state. So we create the subproblem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called **Operator Subgoaling**.

Algorithm for Means-Ends Analysis:

Let's we take Current state as CURRENT and Goal State as GOAL, then following are the steps for the MEA algorithm.

- **Step 1:** Compare CURRENT to GOAL, if there are no differences between both then return Success and Exit.
- **Step 2:** Else, select the most significant difference and reduce it by doing the following steps until the success or failure occurs.

a. Select a new operator O which is applicable for the current difference, and if there is no such operator, then signal failure.

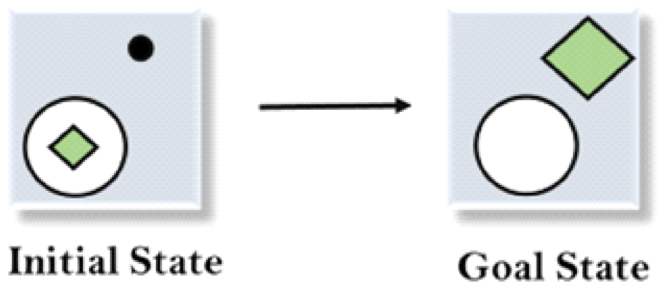
 b. Attempt to apply operator O to CURRENT. Make a description of two states.
 i) O-Start, a state in which O?s preconditions are satisfied.
 ii) O-Result, the state that would result if O were applied In O-start.

 c. If
 (First-Part <------ MEA (CURRENT, O-START)
 And
 (LAST-Part <----- MEA (O-Result, GOAL), are successful, then signal Success and return the result of combining FIRST-PART, O, and LAST-PART.

The above-discussed algorithm is more suitable for a simple problem and not adequate for solving complex problems.

Example of Mean-Ends Analysis:

Let's take an example where we know the initial state and goal state as given below. In this problem, we need to get the goal state by finding differences between the initial state and goal state and applying operators.

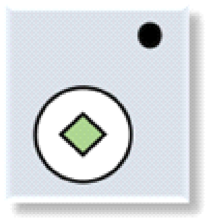


Solution:

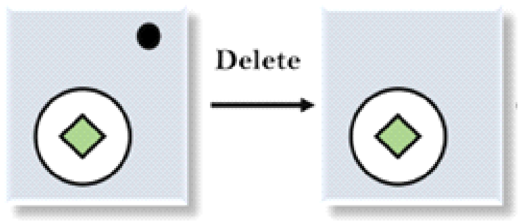
To solve the above problem, we will first find the differences between initial states and goal states, and for each difference, we will generate a new state and will apply the operators. The operators we have for this problem are:

- **Move**
- **Delete**
- **Expand**

1. Evaluating the initial state: In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.

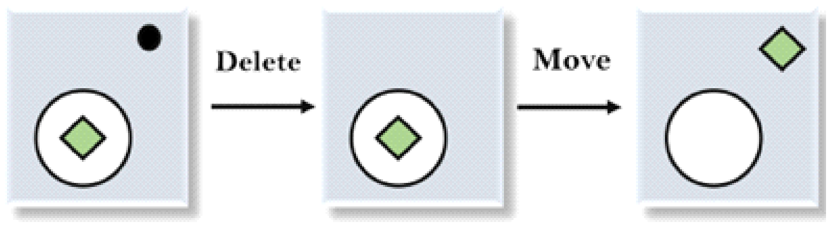

Initial state

2. Applying Delete operator: As we can check the first difference is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the **Delete operator** to remove this dot.



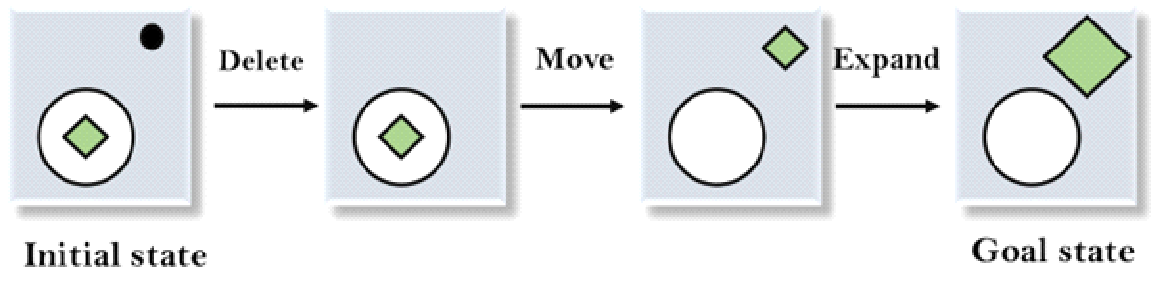
Initial state

3. Applying Move Operator: After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the square is outside the circle, so, we will apply the **Move Operator**.



Initial state

4. Applying Expand Operator: Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the square, so, we will apply **Expand operator**, and finally, it will generate the goal state.



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:

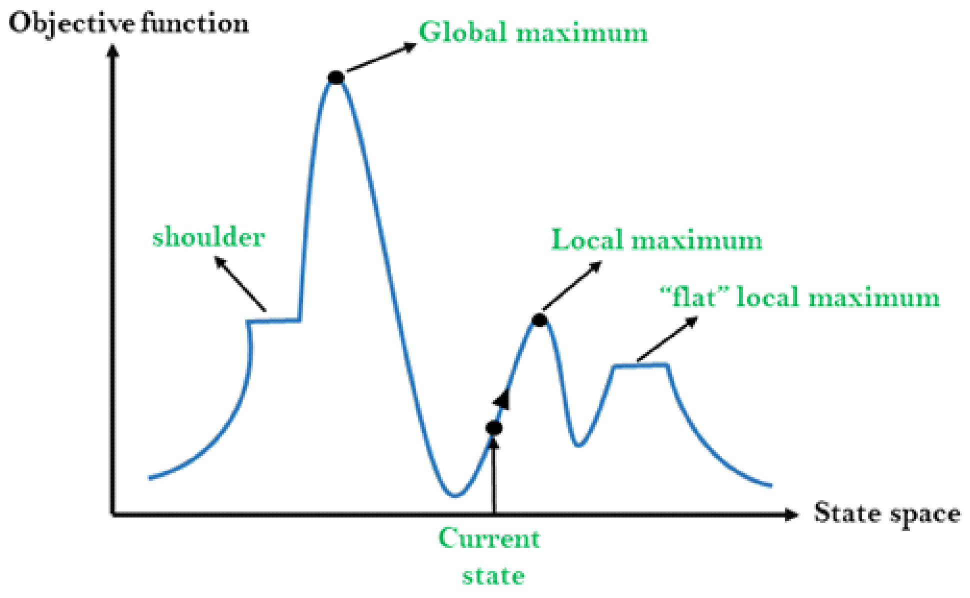
Following are some main features of Hill Climbing Algorithm:

- **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.

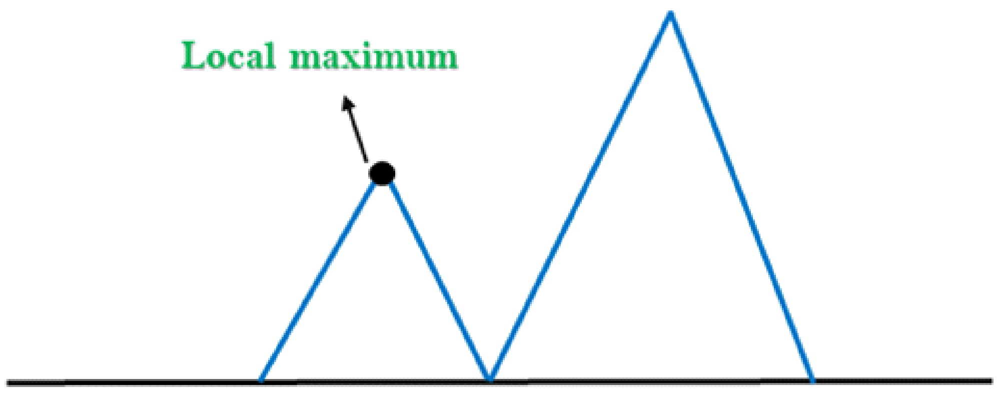
Flat local maximum: It is a flat space in the landscape where all the neighbor states of current states have the same value.

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

Problems in Hill Climbing Algorithm:

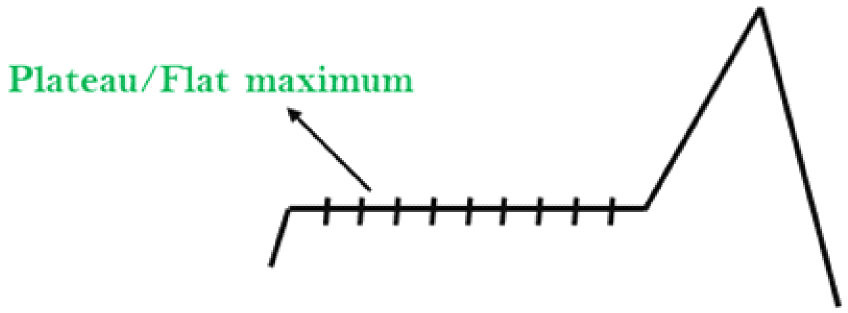
1. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

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.



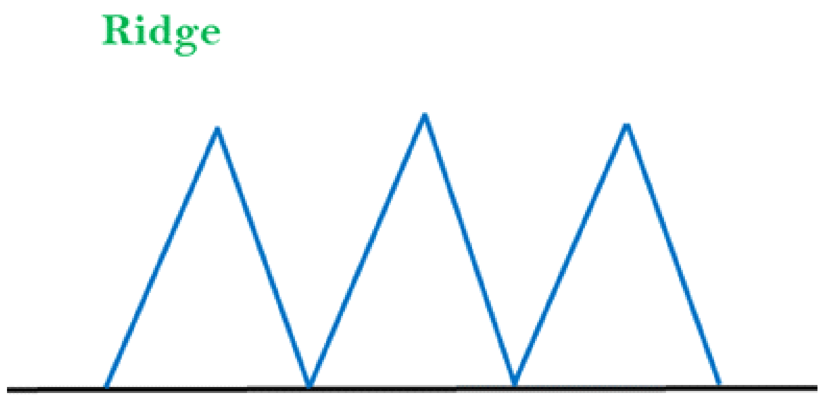
2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

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.



Types of Hill Climbing Algorithm in Artificial Intelligence
Here we discuss the types of a hill-climbing algorithm in artificial intelligence:

Types of Hill Climbing Algorithm:

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

1. Simple Hill Climbing

It is the simplest form of the Hill Climbing Algorithm. It only takes into account the neighboring node for its operation. If the neighboring node is better than the current node then it sets the neighbor node as the current node. The algorithm checks only one neighbor at a time. Following are a few of the key feature of the Simple Hill Climbing Algorithm

- Since it needs low computation power, it consumes lesser time
- The algorithm results in sub-optimal solutions and at times the solution is not guaranteed

Algorithm

1. Examine the current state, Return success if it is a goal state

2. Continue the Loop until a new solution is found or no operators are left to apply

3. Apply the operator to the node in the current state

4. Check for the new state

- If Current State = Goal State, Return success and exit
- Else if New state is better than current state then Goto New state
- Else return to step 2

5. Exit

2. Steepest-Ascent Hill Climbing

Steepest-Ascent hill climbing is an advanced form of simple Hill Climbing Algorithm. It runs through all the nearest neighbor nodes and selects the node which is nearest to the goal state. The algorithm requires more computation power than Simple Hill Climbing Algorithm as it searches through multiple neighbors at once.

Algorithm

1. Examine the current state, Return success if it is a goal state

2. Continue the Loop until a new solution is found or no operators are left to apply

Let ‘Temp’ be a state such that any successor of the current state will have a higher value for the objective function. For all operators that can be applied to the current state

- Apply the operator to create a new state
- Examine new state
- If Current State = Goal State, Return success and exit
- Else if New state is better than Temp then set this state as Temp

- If Temp is better than Current State set Current state to Target

3. Stochastic Hill Climbing

Stochastic Hill Climbing doesn’t look at all its neighboring nodes to check if it is better than the current node instead, it randomly selects one neighboring node, and based on the pre-defined criteria it decides whether to go to the neighboring node or select an alternate node.

Advantage of Hill Climbing Algorithm in Artificial Intelligence

Advantage of Hill Climbing Algorithm in Artificial Intelligence is given below:

- Hill Climbing is very useful in routing-related problems like Travelling Salesmen Problem, Job Scheduling, Chip Designing, and Portfolio Management
- It is good in solving the optimization problem while using only limited computation power
- It is more efficient than other search algorithms

Hill Climbing Algorithm is a very widely used algorithm for Optimization related problems as it gives decent solutions to computationally challenging problems. It has certain drawbacks associated with it like its Local Minima, Ridge, and Plateau problem which can be solved by using some advanced algorithm.

Applications of hill climbing algorithm

The hill-climbing algorithm can be applied in the following areas:

Marketing

A hill-climbing algorithm can help a marketing manager to develop the best marketing plans. This algorithm is widely used in solving Traveling-Salesman problems. It can help by optimizing the distance covered and improving the travel time of sales team members. The algorithm helps establish the local minima efficiently.

Robotics

Hill climbing is useful in the effective operation of robotics. It enhances the coordination of different systems and components in robots.

Job Scheduling

The hill climbing algorithm has also been applied in job scheduling. This is a process in which system resources are allocated to different tasks within a computer system. Job scheduling is achieved through the migration of jobs from one node to a neighboring node. A hill-climbing technique helps establish the right migration route.

Cryptarithmetic Problem in AI Or **Constraint satisfaction problem**

Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement either with alphabets or other symbols. In **cryptarithmetic problem,** the digits (0-9) get substituted by some possible alphabets or symbols. The task in cryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct. We can perform all the arithmetic operations on a given cryptarithmetic problem.

The rules or constraints on a cryptarithmetic problem are as follows:

- There should be a unique digit to be replaced with a unique alphabet.
- The result should satisfy the predefined arithmetic rules, i.e., 2+2 =4, nothing else.
- Digits should be from **0-9** only.
- There should be only one carry forward, while performing the addition operation on a problem.
- The problem can be solved from both sides, i.e., **lefthand side (L.H.S), or righthand side (R.H.S)**

Let's understand the cryptarithmetic problem as well its constraints better with the help of an example:

- Given a cryptarithmetic problem, i.e.,

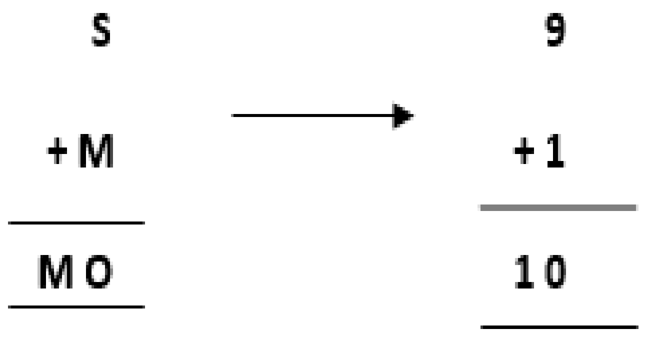
S E N D + M O R E = M O N E Y

```
  SEND
 +MORE
------
 MONEY
------
```

In this example, add both terms **S E N D** and **M O R E** to bring **M O N E Y** as a result.

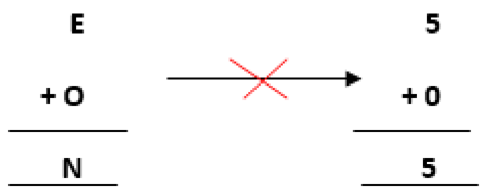
Follow the below steps to understand the given problem by breaking it into its subparts:

- Starting from the left hand side (L.H.S) , the terms are **S** and **M**. Assign a digit which could give a satisfactory result. Let's assign **S->9** and **M->1**.

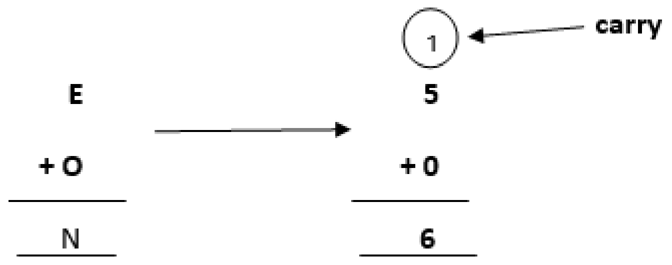


Hence, we get a satisfactory result by adding up the terms and got an assignment for **O** as **O->0** as well.

- Now, move ahead to the next terms **E** and **O** to get **N** as its output.

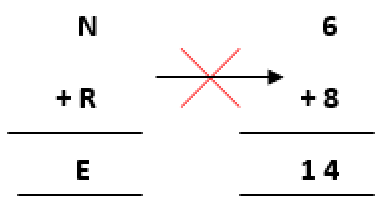


Adding E and O, which means 5+0=0, which is not possible because according to cryptarithmetic constraints, we cannot assign the same digit to two letters. So, we need to think more and assign some other value.



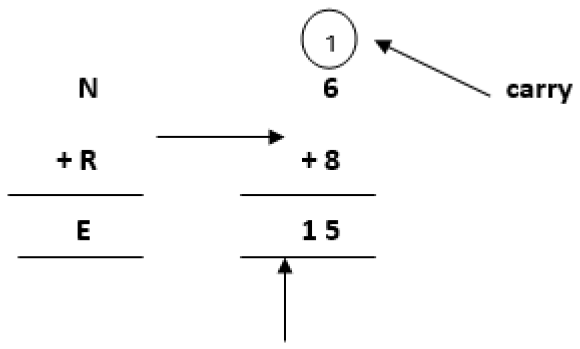
Note: When we will solve further, we will get one carry, so after applying it, the answer will be satisfied.

- Further, adding the next two terms **N** and **R** we get,



But, we have already assigned **E->5**. Thus, the above result does not satisfy the values
because we are getting a different value for **E.** So, we need to think more.

Again, after solving the whole problem, we will get a carryover on this term, so our answer will be satisfied.

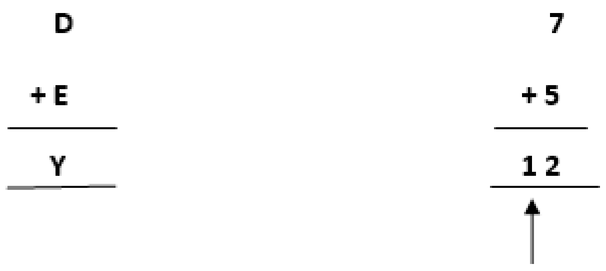


where 1 will be carry forward to the above term

Let’s move ahead.

- Again, on adding the last two terms, i.e., the

  rightmost terms **D** and **E**, we get **Y** as its result.



where 1 will be carry forward to the above term

- Keeping all the constraints in mind, the final resultant is as follows:

```
  S E N D
+ M O R E
---------
M O N E Y
---------
```

- Below is the representation of the assignment of the digits to the alphabets.

S	9
E	5
N	6
D	7
M	1
O	0
R	8
y	2

More examples of cryptarithmatic problems can be:

1.

$$\begin{array}{r} \text{B A S E} \\ \text{+ B A L L} \\ \hline \text{G A M E S} \\ \hline \end{array} \longrightarrow$$

B	7
A	4
S	8
E	3
L	5
G	1
M	9

2.

$$\begin{array}{r} \text{Y O U R} \\ \text{+ Y O U} \\ \hline \text{H E A R T} \\ \hline \end{array} \longrightarrow$$

Y	9
O	4
U	2
R	6
H	1
E	0
A	3
T	8

----------------THE END----------------