Unit-2 Problem Solving by Searching

Problem-solving agent

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. Here are some key characteristics and components of a problem-solving agent:

1. **Perception**: Problem-solving agents typically have the ability to perceive or sense their environment. They can gather information about the current state of the world, often through sensors, cameras, or other data sources.
2. **Knowledge Base**: These agents often possess some form of knowledge or representation of the problem domain. This knowledge can be encoded in various ways, such as rules, facts, or models, depending on the specific problem.
3. **Reasoning**: Problem-solving agents employ reasoning mechanisms to make decisions and select actions based on their perception and knowledge. This involves processing information, making inferences, and selecting the best course of action.
4. **Planning**: For many complex problems, problem-solving agents engage in planning. They consider different sequences of actions to achieve their goals and decide on the most suitable action plan.
5. **Actuation**: After determining the best course of action, problem-solving agents take actions to interact with their environment. This can involve physical actions in the case of robotics or making decisions in more abstract problem-solving domains.
6. **Feedback**: Problem-solving agents often receive feedback from their environment, which they use to adjust their actions and refine their problem-solving strategies. This feedback loop helps them adapt to changing conditions and improve their performance.
7. **Learning**: Some problem-solving agents incorporate machine learning techniques to improve their performance over time.

They can learn from experience, adapt their strategies, and become more efficient at solving similar problems in the future. Problem-solving agents can vary greatly in complexity, from simple algorithms that solve straightforward puzzles to highly sophisticated AI systems that tackle complex, real-world problems. The design and implementation of problem-solving agents depend on the specific problem domain and the goals of the AI application.

9 Real World Problems Effectively Solved by the AI

1. Healthcare:

"Good health is the first condition of happiness." When you are happy, you can make most things work. There are no questions as to why healthcare is one of the most significant industries across the world.

So, how can AI solve healthcare problems? AI can refine huge data in record time supporting investigators to save significant data faster and emphasize their research instead of data refining and accumulation. Probing is a significant part of healthcare as it allows us to determine several health problems, identify treatment methods, research cures, and others. Using AI, doctors and researchers can determine the effectiveness and outcome of a specific drug treatment for an individual before suggesting the medicine.

2. Wildlife Conservation:

AI's ability to examine ample information at once can help conservationists to learn about wildlife protection. For example, by tracking animals' mobility, they can understand their patterns and movements. AI tracks wildlife habitats and foretells the extinction of endangered animal species. It helps to detect and prevent wildlife poaching. AI further provides information regarding the impact of climate change on wildlife and decreases its effect by developing an accurate plan. It helps in evaluating the population of species and observing transmission. AI significantly assists in preventing illegal animal trade on social media.

3. Learning & Training:

Think about chatting with a competent teaching assistant during your academic years. They are extremely helpful and useful, but you have never met the individual. This is what something similar happened to students at Georgia Tech when they identified that their teaching assistant was a robot.

People are significantly the same. The only difference is they hold information differently and at varying speeds. Using AI helps to personalize teaching mechanisms that help professors and students. For instance, professors can understand how to approach a specific child's learning curve. Also, using AI for solving this real-world problem can help AI teachers to tutor the child without getting tired as they sound and look like individuals.

4. Transportation:

AI further helps in solving real-world problems like transportation. By using machine learning – powered route optimization software, AI helps to develop an optimized route to locations for several uses whilst reducing the costs of roads and the number of vehicles. AI uses the following step to do so:

- The customer chooses the departure place, the position of destination, and a particular time of the day.
- Now, AI logistics evaluates several attributes such as weather conditions, traffic reports, and patterns, and produces the finest possible route.
- Logistics often depends upon historical traffic information to analyze the best day of the month and a specific time of the day when the customer can enjoy the ride and go out on the road.

5. Hiring:

HR department is required to go through plethora of resumes. Now, imagine if HR personnel can refine the list of candidates they are looking for within seconds. That's where AI comes to the rescue. AI helps the HR team to take up the monotonous task of searching resumes and

provides you with the most relevant data. This offers HR employees spare time for interviewing more efficiently, collection needs insights from the data throughout the process.

Predictive analytics further helps to understand trends between those who got hired and applied. It helps HR personnel to arrange significant reports which signify any identifiable structure.

6. Renewable Energy Sector:

AI technologies provide insight by analyzing the data for the grid managers for effective control operations. It provides flexibility to the energy providers to intelligently manage the supply and demand chain. Generative AI helps with the amalgamation of microgrids and handling distributed energy. It plays a crucial role in resolving congestion and quality problems. Also, helps to comprehend energy consumption standards and find out the energy leakage and health devices. By using AI, you can also analyze the data linked with energy collection and offer details about energy consumption. This helps to standardize the occurring services and develop new service models.

7. Research and Development:

AI is advancing the research and development industry by having a significant impact on the following:

- Smart platforms
- Creation of individual learning paths
- Behavioral analysis
- Knowledge management
- Intelligent monitoring of infrastructures
- Technical and medical diagnostic systems
- Collection, and processing of massive amounts of data, and much more

Researchers are always looking for opportunities, and patterns to create models, uncover efficiencies, and more. Regardless of the industry, AI helps people to get insight into massive amounts of data to accelerate data depending on decision-making and improving informed

frameworks. Furthermore, AI helps to overcome problems like human emissions as researchers need data on the atmospheric methane cycle.

8. Logistics and Operations:

Some of the benefits are:

- **Back office:** AI enables organizations to meet the full end-to-end automation of back-office processes. Hence, it enhances Robotic Process Automation (RPA) conditions or even finds other alternatives. Where RPA works through logic and commands, the AI continues to enhance the process by itself.
- **Vehicle Telematics:** AI helps to identify delivery misuse and mistakes in vehicles. It is a significant feature for reducing possible human attribute issues.
- **Driver profiling and safety:** This enables the drivers to reassess their actions and improve their driving abilities. This enhances both satisfaction and effectiveness from the perspective of driving.
- **Demand prediction:** AI assists in decreasing the stockout count which is one of the most frustrating problems for any logistics-linked organization and its customers.
- **Route planning:** It decreases the time invested in every route or delivery such that the performed deliveries increase correspondingly.
- **Automated warehouses:** Automation of warehouses through AI enables the development of a more accessible, and safer working surrounding that can be a hiring merit in the name of labor shortages.
- **Delivery automation:** It enables standardizing and choosing all the delivery categories like the distribution of items per vehicle. This leads to a reduced time of delivery as the fleet spends less time on every order.

9. Marketing and Sales:

Several sales and marketing problems can be solved by changing their AI customer experience in the following ways:

- Producing more occasions for your business: By interacting with the visitors, collecting their information such as name and phone number, or by helping the marketing department in real-time to qualify leads.
- As soon as the qualified leads are gathered in the CRM unit, the sales department will see this information. AI assists in converting information into meaningful insights by offering them with the finest customer fit, predictive scoring, and taking appropriate actions to convert leads.

Challenges

Using artificial intelligence produces several challenges, but the most significant problem is how we are required to keep the systems secure. Statistics depend on information, so any reform to that information will reform the outcomes and behaviour.

Unquestionably, AI has so much to offer without destroying the human lifestyle, only otherwise we misuse it.

Toy problem

A toy problem in artificial intelligence is a simplified or abstract problem that is used to help researchers explore and understand the basic concepts and principles of AI algorithms and techniques. Toy problems are often used in AI research as a way to test and validate new ideas, algorithms, and models.

Toy problems are different from real-world problems because they require solutions and are not just descriptions. They can illustrate various problem-solving methods and are useful for:

- Testing and demonstrating methodologies
- Comparing the performance of different algorithms
- Explaining a particular, more general, problem-solving technique

Some examples of toy problems include: Chess game, Sliding-block puzzles, N-Queens problem, Missionaries and cannibals' problem, and Tic-Tac-Toe game.

Informed search strategies

What are informed search strategies in AI?
The algorithms of an informed search contain information regarding the goal state. It helps an AI make more efficient and accurate searches. A function obtains this data/info to estimate the closeness of a state to its goal in the system. For example, Graph Search and Greedy Search.

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.

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.

1. 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.

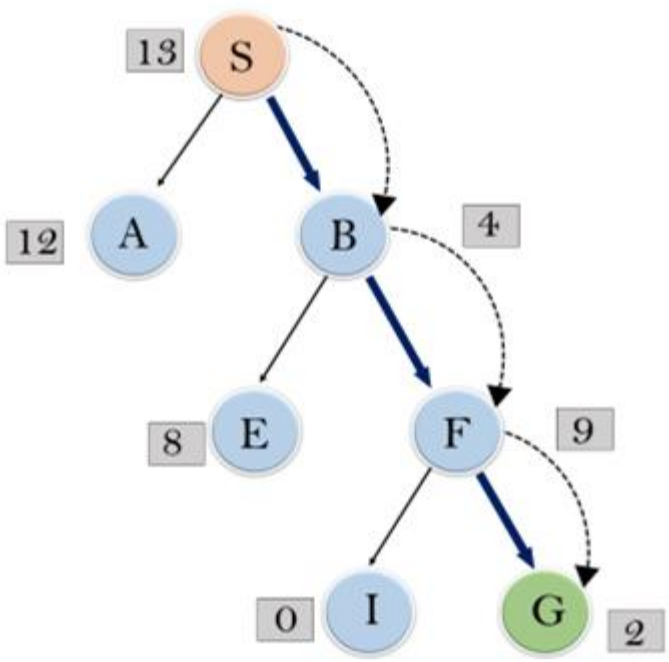
Disadvantages:

- 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.

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



Expand the nodes of S and put in the CLOSED list

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

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

Iteration 2: Open [E, F, A], Closed [S, B]
: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]
: Open [I, E, A], Closed [S, B, F, G]

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

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

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

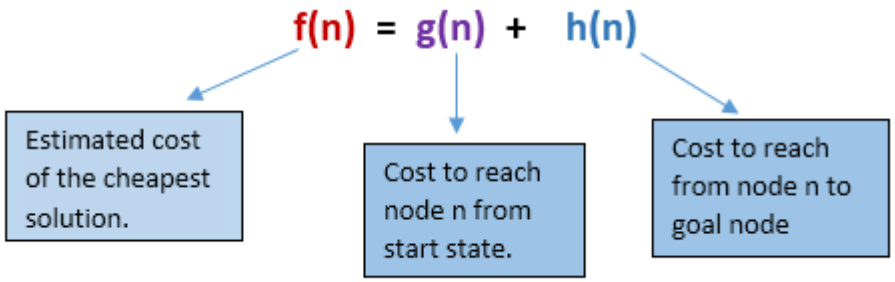
Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

2.) A* Search Algorithm:

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

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



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.

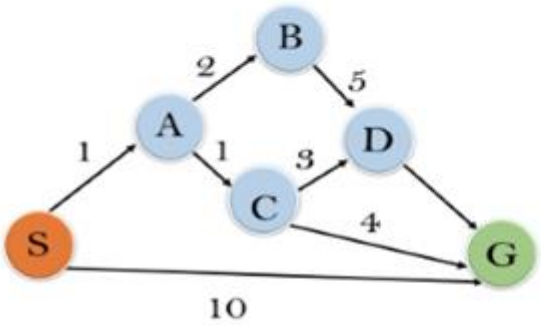
Disadvantages:

- 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:

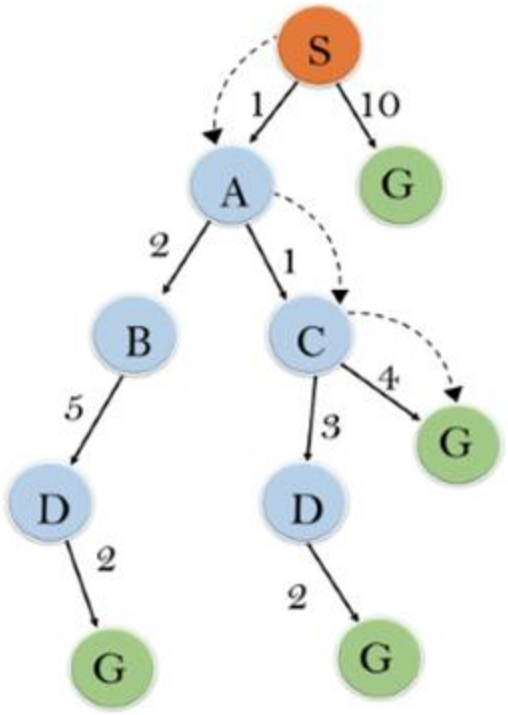
In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.

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)**

Uninformed Search Strategies:

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.

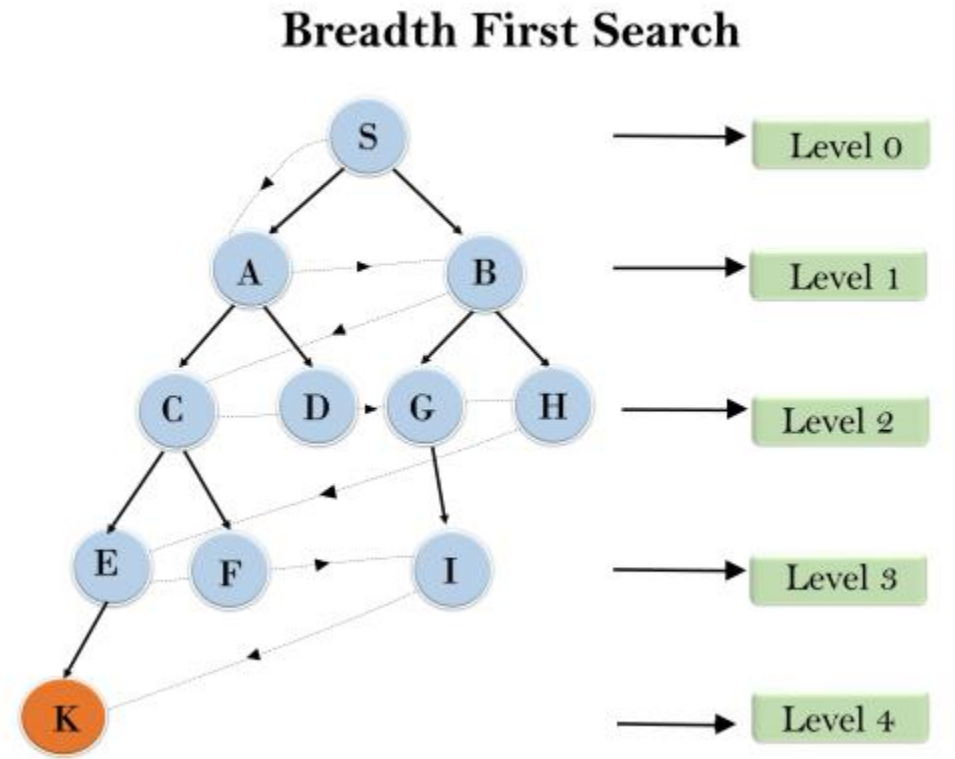
Disadvantages:

- 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
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:

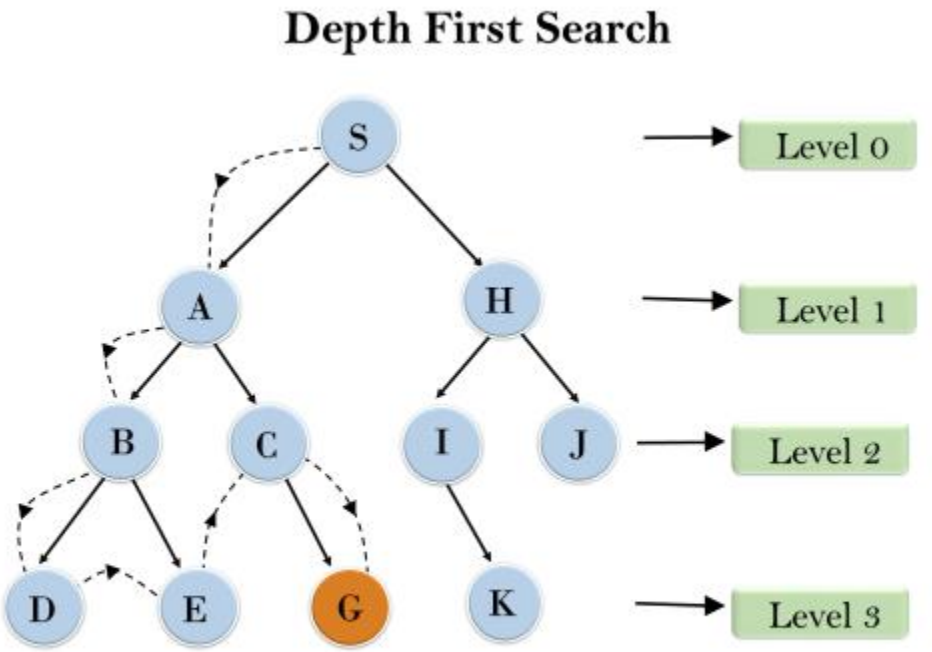
- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

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.

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:

1. $h(n) <= h^*(n)$

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.

Searching for Solution:

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.

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:
    a. **Search Space:** Search space represents a set of possible solutions, which a system may have.
    b. **Start State:** It is a state from where agent begins **the search**.
    c. **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.
- **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.
- **Optimal Solution:** If a solution has the lowest cost among all solutions.

Properties of Search Algorithms:

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

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

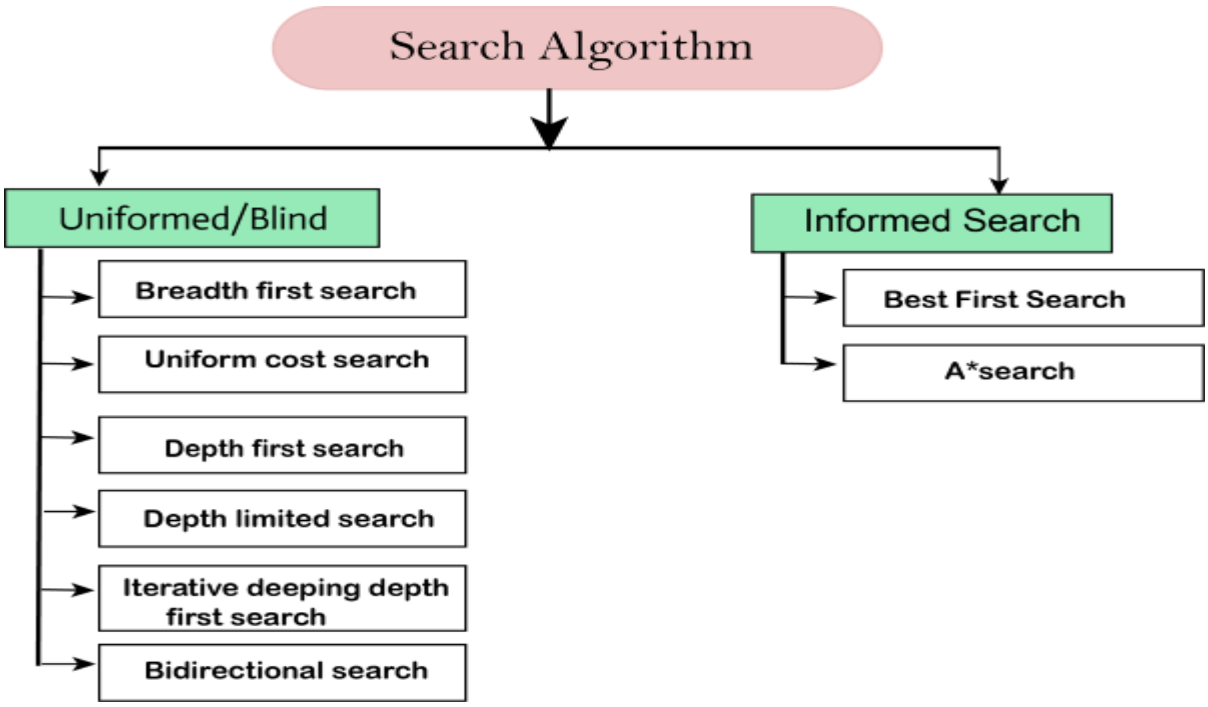
Optimality: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

Time Complexity: Time complexity is a measure of time for an algorithm to complete its task.

Space Complexity: It is the maximum storage space required at any point during the search, as the complexity of the problem.

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

Local Search Algorithms in AI:

Artificial Intelligence (AI) is revolutionizing how we solve complex problems and make decisions. One crucial aspect of AI is local search algorithms, which play a significant role in finding optimal solutions in various domains. In this article, we will delve into the concept of local search in AI, its workings, different algorithms, and its practical applications.

What is Local Search in AI?

Local search in AI refers to a family of optimization algorithms that are used to find the best possible solution within a given search space. Unlike global search methods that explore the entire solution space, local search algorithms focus on making incremental changes to improve a current solution until they reach a locally optimal or satisfactory solution. This approach is useful in situations where the solution space is vast, making an exhaustive search impractical.

Working of a Local Search Algorithm

The basic working principle of a local search algorithm involves the following steps:

- Initialization: Start with an initial solution, which can be generated randomly or through some heuristic method.
- Evaluation: Evaluate the quality of the initial solution using an objective function or a fitness measure. This function quantifies how close the solution is to the desired outcome.
- Neighbour Generation: Generate a set of neighbouring solutions by making minor changes to the current solution. These changes are typically referred to as "moves."

- Selection: Choose one of the neighbouring solutions based on a criterion, such as the improvement in the objective function value. This step determines the direction in which the search proceeds.
- Termination: Continue the process iteratively, moving to the selected neighbouring solution, and repeating steps 2 to 4 until a termination condition is met. This condition could be a maximum number of iterations, reaching a predefined threshold, or finding a satisfactory solution.

Local Search Algorithms

Several local search algorithms are commonly used in AI and optimization problems. Let's explore a few of them:

Let's delve into some of the commonly used local search algorithms:

1. Hill Climbing

Hill climbing is a straightforward local search algorithm that starts with an initial solution and iteratively moves to the best neighbouring solution that improves the objective function. Here's how it works:

- Initialization: Begin with an initial solution, often generated randomly or using a heuristic method.
- Evaluation: Calculate the quality of the initial solution using an objective function or fitness measure.
- Neighbour Generation: Generate neighbouring solutions by making small changes (moves) to the current solution.
- Selection: Choose the neighbouring solution that results in the most significant improvement in the objective function.
- Termination: Continue this process until a termination condition is met (e.g., reaching a maximum number of iterations or finding a satisfactory solution).

Hill climbing has a limitation in that it can get stuck in local optima, which are solutions that are better than their neighbours but not necessarily the best overall solution. To overcome this limitation, variations of hill climbing algorithms have been developed, such as stochastic hill climbing and simulated annealing.