SAPTHAGIRI NPS UNIVERSITY
UNMATCHED EXCELLENCE, UNLIMITED POTENTIAL

1st Semester

Fundamentals of AI and ML (24BTPHY106)

Module-2
Problem-solving - Solving Problems by Searching

Solving Problems by Searching Problem-Solving Agents — Example Problems — Search Algorithms — Uninformed Search Strategies — Informed (Heuristic) Search Strategies —Heuristic Functions

PROBLEM SOLVING BASED ON SEARCHING

Search algorithms are one of the most important areas of Artificial Intelligence. This topic will explain all about the search algorithms in AI.

Problem-solving agents:

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

Search Algorithm Terminologies:

Search: Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:

- **Search Space:** Search space represents a set of possible solutions, which a system may have.
- **Start State:** It is a state from where agent begins **the search**.
- **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.

- **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.
- **Optimal Solution:** If a solution has the lowest cost among all solutions.

Properties of Search 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.

EXAMPLE PROBLEMS:

Real-world problems:

Consider the airline travel problems that must be solved by a travel-planning Web site

States: Each state obviously includes a location (e.g., an airport) and the current time. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their fare bases, and their status as

domestic or international, the state must record extra information about these “historical” aspects.

Initial state: This is specified by the user’s query.

- Actions: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.

- 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, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on

Commercial travel advice systems use a problem formulation of this kind, with many additional complications to handle the byzantine fare structures that airlines impose. A really good system should include contingency plans—such as backup reservations on alternate flights— to the extent that these are justified by the cost and likelihood of failure of the original plan.

1.TRAVELING SALESPERSON PROBLEM

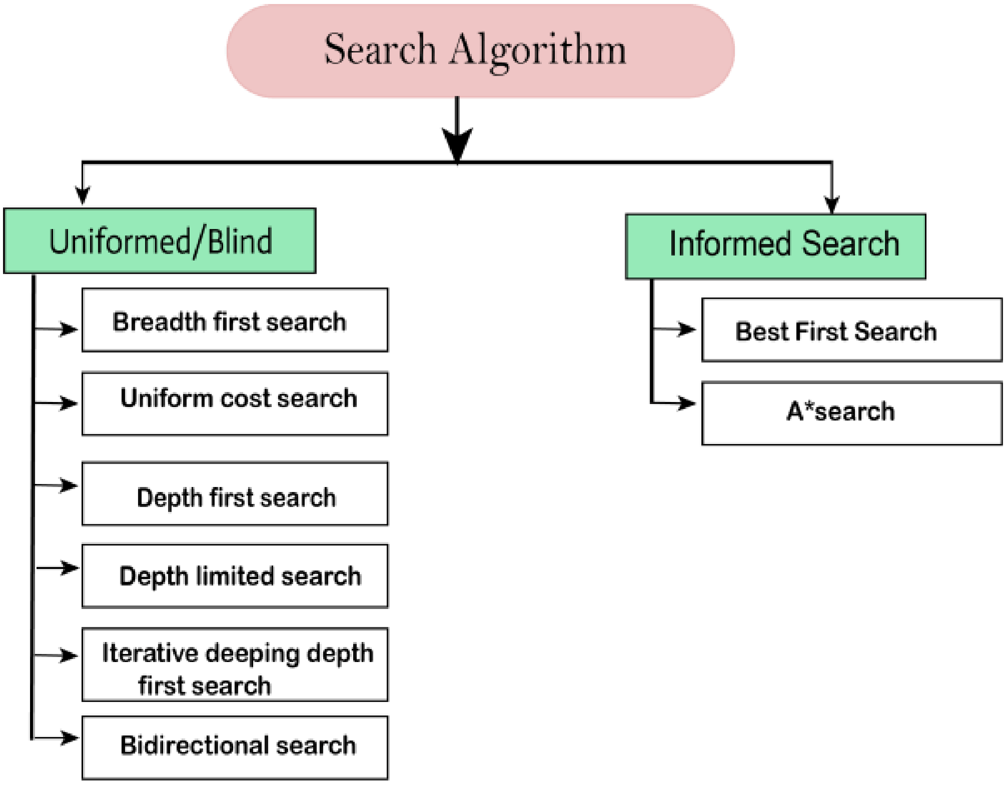
The traveling salesperson problem (TSP) is a touring problem in which each city must be visited exactly once. The aim is to find the shortest tour. The problem is known to be NP-hard, but an enormous amount of effort has been expended to improve the capabilities of TSP algorithms. In addition to planning trips for traveling salespersons, these algorithms have been used for tasks such as planning movements of automatic circuit-board drills and of stocking machines on shop floors.

2. VLSI LAYOUT

A VLSI layout problem requires positioning millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield. The layout problem comes after the logical design phase and is usually split into two parts: cell layout and channel routing. In cell layout, the primitive components of the circuit are grouped into cells, each of which performs some recognized function. Each cell has a 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 the connecting wires to be placed between the cells. Channel routing finds a specific route for each wire through the gaps between the cells

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.



What is the Uninformed Search?

- Uninformed search, also known as blind search, is a search algorithm that explores a problem space without any specific knowledge or information about the problem other than the initial state and the possible actions to take.
- It lacks domain-specific heuristics or prior knowledge about the problem.
- Uninformed search algorithms, such as breadth-first search and depth-first search, systematically explore the search space by applying predefined rules to generate successor states until a goal state is found or the search is exhausted.
- These algorithms are typically less efficient than informed search algorithms but can be useful in certain scenarios or as a basis for more advanced search techniques.

Types of Uninformed Search Algorithms

The different types of uninformed search algorithms used in AI are as follows:

- Depth First Search
- Breadth-First Search
- Depth Limited Search
- Uniform Cost Search

- Iterative Deepening Depth First Search

- Bidirectional Search (if applicable)

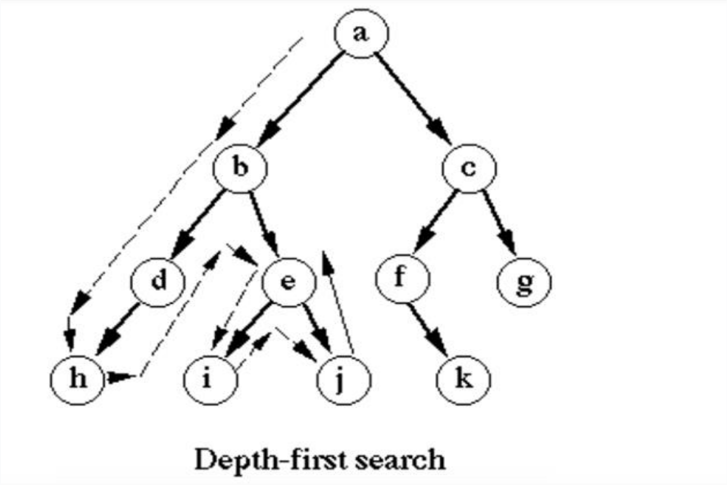
let's get to know the few terms which will be frequently used in the upcoming sections.

- **State**: It provides all the information about the environment.
- **Goal State**: The desired resulting condition in a given problem and the kind of search algorithm we are looking for.
- **Goal Test**: The test to determine whether a particular state is a goal state.
- **Path/Step Cost**: These are integers that represent the cost to move from one node to another node.
- **Space Complexity**: A function describing the amount of space(memory) an algorithm takes in terms of input to the algorithm.
- **Time Complexity**: A function describing the amount of time the algorithm takes in terms of input to the algorithm.
- **Optimal**: Extent of preference of the algorithm

- '**b**' – maximum branching factor in a tree.
- '**d**' – the depth of the least-cost solution.
- '**m**' – maximum depth state space(maybe infinity)

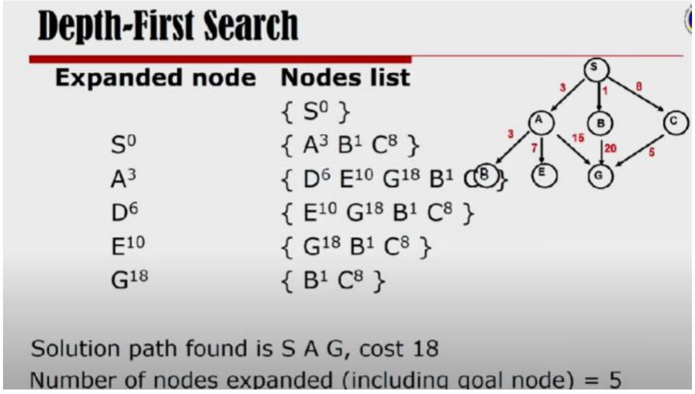
Depth First Search (DFS)

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- It is a search algorithm where the search tree will be traversed from the root node. It will be traversing, searching for a key at the leaf of a particular branch. If the key is not found, the searcher retraces its steps back (backtracking) to the point from where the other branch was left unexplored, and the same procedure is repeated for that other branch.



Depth-first search

- First, the search technique starts from the root node A and then goes to the branch where node B is present (lexicographical order).
- Then it goes to node D because of DFS, and from D, there is only one node to traverse, i.e., node H.
- But after node H does not have any child nodes, we retrace the path in which we traversed earlier and again reach node B, but this time, we traverse through in the untraced path a traverse through node E.
- There are two branches at node E, but let’s traverse node I (lexicographical order) and then retrace the path as we have no further number of nodes after E to traverse.
- Then we traverse node J as it is the untraced branch and then again find we are at the end and retrace the path and reach node B and then we will traverse the untraced branch, i.e., through node C, and repeat the same process. This is called the DFS Algorithm.



Depth-First Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ D^6 E^{10} G^{18} B^1 C^8 }
D^6	{ E^{10} G^{18} B^1 C^8 }
E^{10}	{ G^{18} B^1 C^8 }
G^{18}	{ B^1 C^8 }

Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

Advantages

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

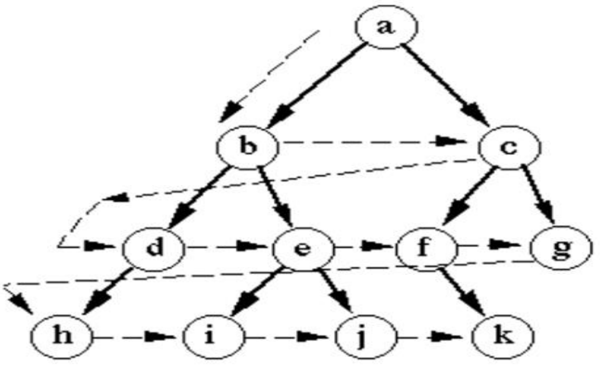
Disadvantages

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

Breadth-First Search (BFS)

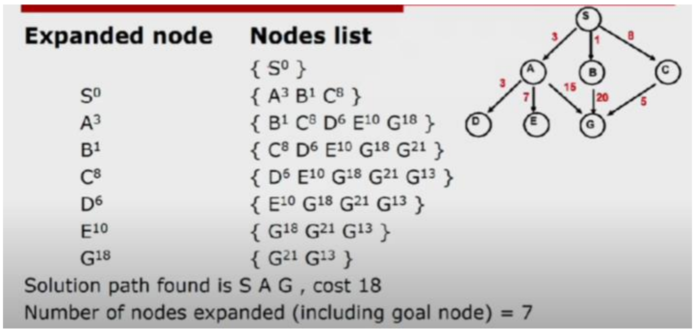
- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before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

This is another graph search algorithm in AI that traverses breadthwise to search for the goal in a tree. It begins searching from the root node and expands the successor node before expanding further along breadthwise and traversing those nodes rather than searching depth-wise.



Breadth-first search

The above figure is an example of a BFS Algorithm. It starts from the root node A and then traverses node B. Till this step, it is the same as DFS. But here, instead of expanding the children of B as in the case of DFS, we expand the other child of A, i.e., node C because of BFS, and then move to the next level and traverse from D to G and then from H to K in this typical example. To traverse here, we have only taken into consideration the lexicographical order. This is how the BFS Algorithm is implemented.



Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ B^1 C^8 D^6 E^{10} G^{18} }
B^1	{ C^8 D^6 E^{10} G^{18} G^{21} }
C^8	{ D^6 E^{10} G^{18} G^{21} G^{13} }
D^6	{ E^{10} G^{18} G^{21} G^{13} }
E^{10}	{ G^{18} G^{21} G^{13} }
G^{18}	{ G^{21} G^{13} }

Solution path found is S A G , cost 18

Number of nodes expanded (including goal node) = 7

Advantages

- BFS will provide a solution if any solution exists.
- If there is more than one solution 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 in memory to expand to the next level.

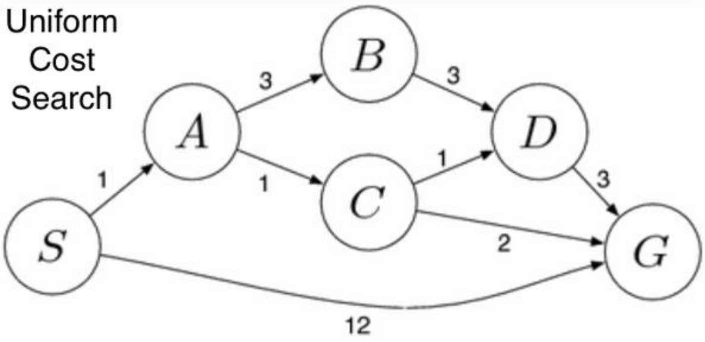
BFS needs lots of time if the solution is far away from the root node.

Uniform Cost Search Algorithm (UCS)

Uniform Cost Search (UCS) is a graph traversal and search algorithm used in the field of artificial intelligence and computer science. UCS is an informed search algorithm that explores a graph by gradually expanding nodes starting from the initial node and moving towards the goal node while considering the cost associated with each edge or step.

This algorithm is mainly used when the step costs are not the same, but we need the optimal solution to the goal state.

It does not go depth or breadth. It searches for the next node with the lowest cost, and in the case of the same path cost, let’s consider lexicographical order in our case.



- In the above figure, consider S to be the start node and G to be the goal state.
- From node S we look for a node to expand, and we have nodes A and G, but since it's a uniform cost search, it's expanding the node with the lowest step cost, so node A becomes the successor rather than our required goal node G.
- From A we look at its children nodes B and C. Since C has the lowest step cost, it traverses through node C.
- Then we look at the successors of C, i.e., D and G. Since the cost to D is low, we expand along with node D.
- Since D has only one child G which is our required goal state
- we finally reach the goal state D by implementing UFS Algorithm.
- If we have traversed this way, definitely our total path cost from S to G is just 6 even after traversing through many nodes rather than going to G directly where the cost is 12 and 6<<12(in terms of step cost). But this may not work with all cases.

Advantages

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

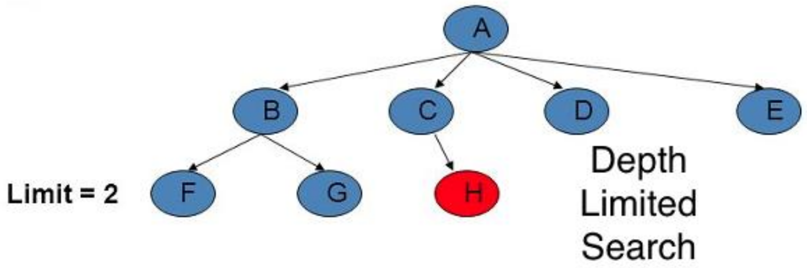
Disadvantages

- It does not care about the number of steps or finding the shortest path involved in the search problem, and it is only concerned about path cost. This algorithm may be stuck in an infinite loop.

Depth Limited Search (DLS)

DLS is an uninformed search algorithm. This is similar to DFS but differs only in a few ways. The sad failure of DFS is alleviated by supplying a depth-first search with a predetermined depth limit. That is, nodes at depth are treated as if they have no successors. This approach is called a depth-limited search. The depth limit solves the infinite-path problem. Depth-limited search can be halted in two cases:

1. Standard Failure Value (SFV): The SFV tells that there is no solution to the problem.
2. Cutoff Failure Value (CFV): The Cutoff Failure Value tells that there is no solution within the given depth limit.



The above figure illustrates the implementation of the DLS algorithm. Node A is at Limit = 0, followed by nodes B, C, D, and E at Limit = 1 and nodes F, G, and H at Limit = 2. Our start state is considered to be node A, and our goal state is node H. To reach node H, we apply DLS. So in the first case, let's set our limit to 0 and search for the goal.

Since limit 0, the algorithm will assume that there are no children after limit 0 even if nodes exist further. Now, if we implement it, we will traverse only node A as there is only one node in limit 0, which is basically our goal state. If we use SFV, it says there is no solution to the problem at limit 0, whereas LCV says there is no solution for the problem until the set depth limit. Since we could not find the goal, let's increase our limit to 1 and apply DFS till limit 1, even though there are further nodes after limit 1. But those nodes aren't expanded as we have set our limit as 1.

Hence nodes A, followed by B, C, D, and E, are expanded in the mentioned order. As in our first case, if we use SFV, it says there is no solution to the problem at limit 1, whereas LCV says there is no solution for the problem until the set depth limit 1. Hence we again increase our limit from 1 to 2 in the notion to find the goal.

Till limit 2, DFS will be implemented from our start node A and its children B, C, D, and E. Then from E, it moves to F, similarly backtracks the path, and explores the unexplored branch where node G is present. It then retraces the path and explores the child of C, i.e., node H, and then we finally reach our goal by applying DLS Algorithm. Suppose we have further successors of node F but only the nodes till limit 2 will be explored as we have limited the depth and have reached the goal state

Advantages

- Depth-limited search is Memory efficient.

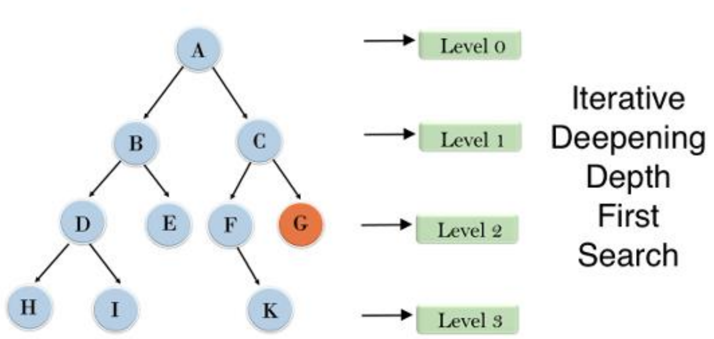
Disadvantages

- The DLS has disadvantages of completeness and is not optimal if it has more than one goal state.

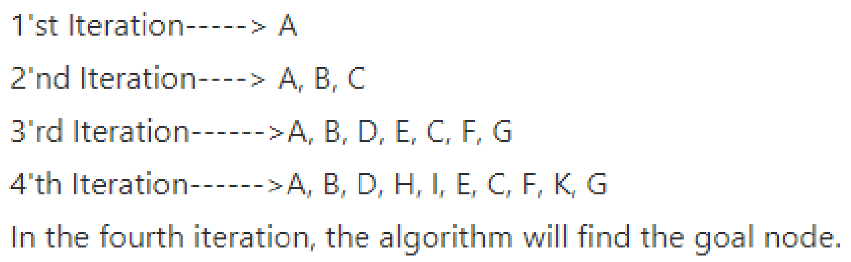
Iterative deepening depth-first Search:

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.



In the above figure, let’s consider the goal node to be G and the start state to be A. We perform our IDDFS from node A. In the first iteration, it traverses only node A at level 0. Since the goal is not reached, we expand our nodes, go to the next level, i.e., 1 and move to the next iteration. Then in the next iteration, we traverse the node A, B, and C. Even in this iteration, our goal state is not reached, so we expand the node to the next level, i.e., 2, and the nodes are traversed from the start node or the previous iteration and expand the nodes A, B, C, and D, E, F, G. Even though the goal node is traversed, we go through for the next iteration, and the remaining nodes A, B, D, H, I, E, C, F, K, and G(BFS & DFS) too are explored, and we find the goal state in this iteration. This is the implementation of the IDDFS Algorithm


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.

Advantages

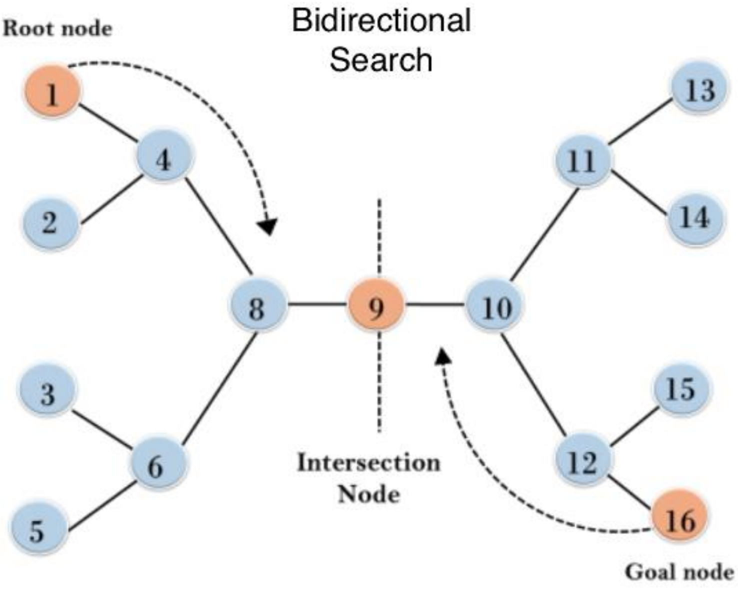
- It combines the benefits of BFS and DFS search algorithms in terms of fast search and memory efficiency.

Disadvantages

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

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.



This figure provides a clear-cut idea of how BS is executed. We have node 1 as the start/root node and node 16 as the goal node. The algorithm divides the search tree into two sub-trees. So from the start of node 1, we do a forward search, and at the same time, we do a backward search from goal node 16. The forward search traverses nodes 1, 4, 8, and 9, whereas the backward search traverses through nodes 16, 12, 10, and 9. We see that both forward and backward search meets at node 9, called the intersection node. So the total path traced by forwarding search and the path traced by backward search is the optimal solution. This is how the BS Algorithm is implemented.

Advantages

- Since BS uses various techniques like DFS, BFS, DLS, etc., it is efficient and requires less memory.

Disadvantages

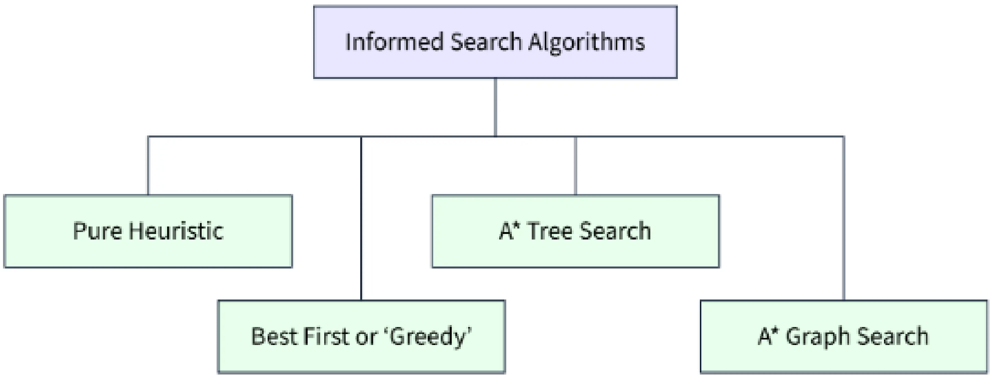
- Implementation of the bidirectional search tree is difficult.

Informed Search

Informed Search algorithms play a significant role in artificial intelligence. These algorithms enable more effective and efficient search using heuristic functions to direct the search process towards a goal state

Informed search algorithms are a type of search algorithm that uses heuristic functions to guide the search process. For example, a heuristic function calculates the cost of moving from a starting state to a goal state. By employing this assessment, informed search algorithms can select search paths more likely to lead to the desired state. As a result, the search process is improved, making it quicker and more accurate for AI systems to make decisions.

Types of informed Search Algorithms



Heuristics function

A heuristic function calculates the cost of moving from a starting state to a goal state. This function directs the search process in intelligent algorithms like A* search and the best first search in artificial intelligence. The algorithm can prioritize search paths that are more likely to lead to the goal state by using the heuristic function, which gives an estimate of how close a particular state is to the goal state.

Greedy Best first search algorithm

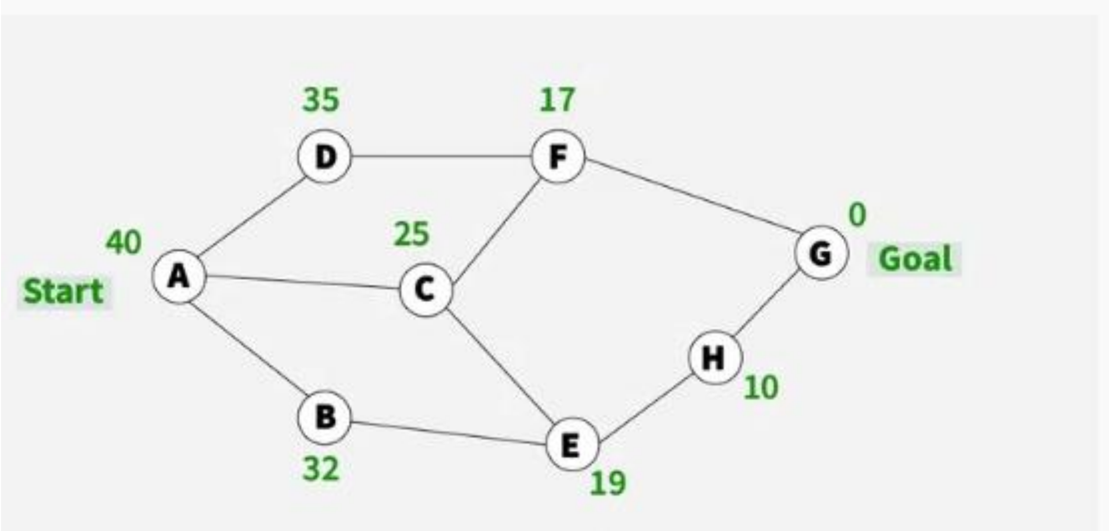
Greedy Best-First Search is an AI search algorithm that attempts to find the most promising path from a given starting point to a goal. It prioritizes paths that appear to be the most promising, regardless of whether or not they are actually the shortest path. The algorithm works by evaluating the cost of each possible path and then expanding the path with the lowest cost. This process is repeated until the goal is reached.

How Greedy Best-First Search Works?

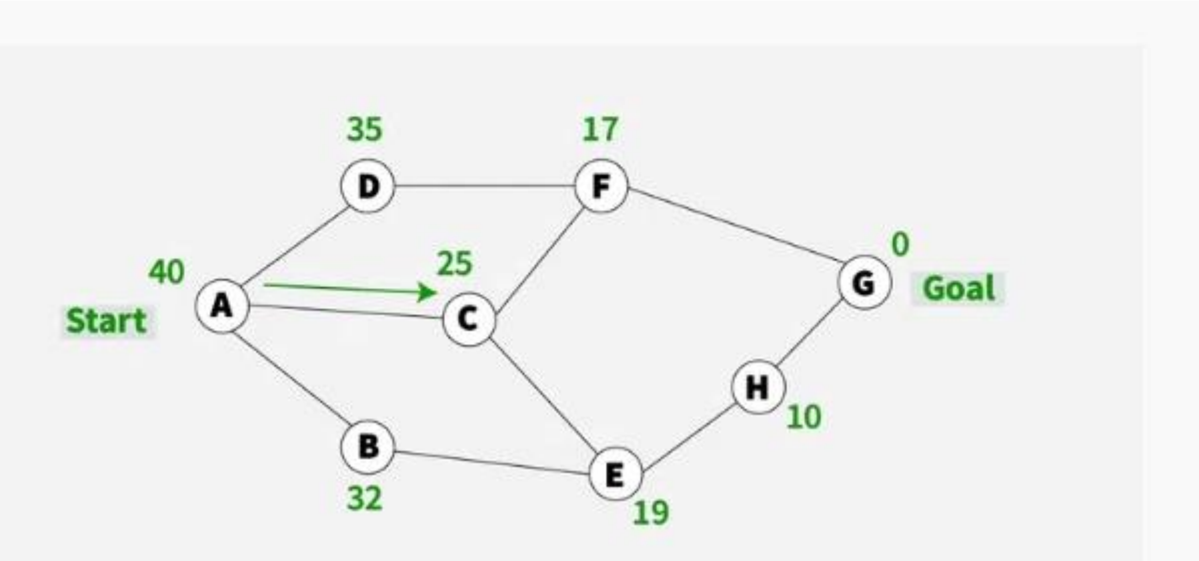
- Greedy Best-First Search works by evaluating the cost of each possible path and then expanding the path with the lowest cost. This process is repeated until the goal is reached.
- The algorithm uses a heuristic function to determine which path is the most promising.
- The heuristic function takes into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

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

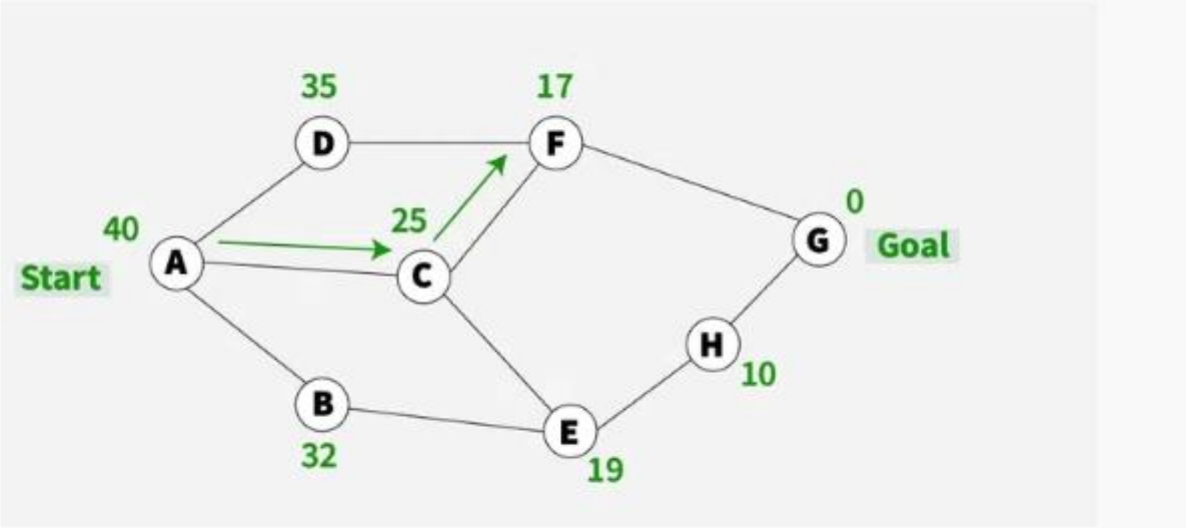
The values represent the heuristic value of reaching the goal node G from current node



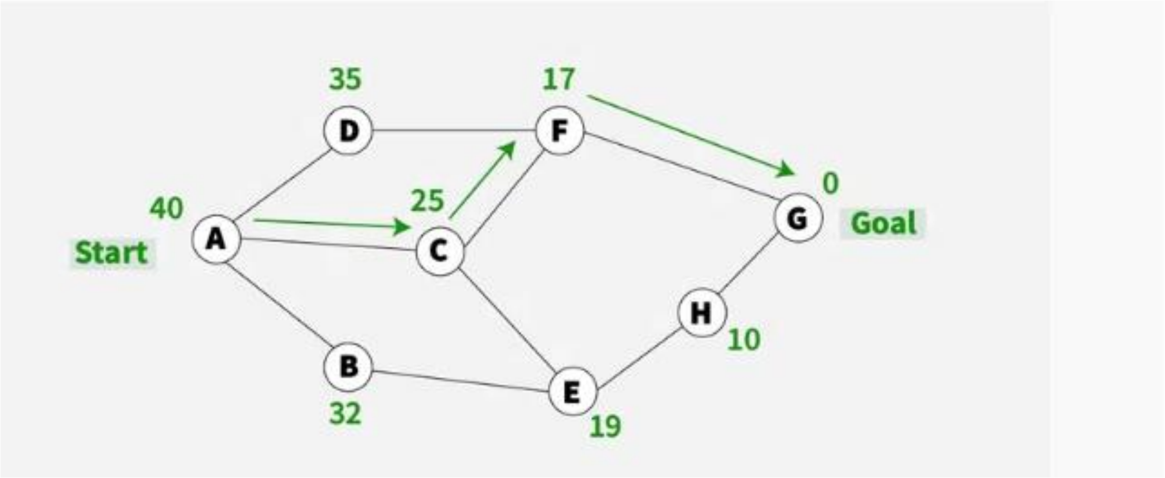
1) We are starting from A , so from A there are direct path to node B(with heuristics value of 32) , from A to C (with heuristics value of 25) and from A to D(with heuristics value of 35) .
2) So as per best first search algorithm choose the path with lowest heuristics value , currently C has lowest value among above node . So we will go from A to C.



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



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



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

Advantages of Greedy Best-First Search:

- Simple and Easy to Implement
- Fast and Efficient
- Low Memory Requirements
- Flexible
- Efficiency

Disadvantages of Greedy Best-First Search:

- **Inaccurate Results**
- **Local Optima:** Greedy Best-First Search can get stuck in local optima, meaning that the path chosen may not be the best possible path.
- **Heuristic Function:** Greedy Best-First Search requires a heuristic function in order to work, which adds complexity to the algorithm

A* algorithm

The A* algorithm is used to find the shortest path between nodes on a graph. It uses two lists - OPEN and CLOSED - to track nodes. The algorithm calculates f(n)=g(n)+h(n) to determine which node to expand next, where g(n) is the cost to reach node n from the starting node and h(n) is a heuristic estimate of the cost to reach the goal from n.

Here are core components of A* algorithm in AI

Open list: The open list is a collection of nodes that are candidates for evaluation. Initially, it contains only the starting node. As the algorithm progresses, nodes are added or removed from the open set based on their estimated total cost (usually denoted as "f-score"). The node with the lowest f-score is selected for evaluation next.

Closed list: The closed list contains nodes that have already been evaluated. Once a node is evaluated, it is moved from the open list to the closed list. This prevents revisiting nodes and ensures that the algorithm explores the graph efficiently.

Cost Function: The A* algorithm uses a cost function that assigns a cost (often referred to as "g(n)") to each node based on the cost of reaching that node from the starting point. Additionally, it calculates a heuristic cost estimate (often referred to as "h(n)") from that node to the goal node. The f-score of a node is the sum of its actual cost (g(n)) and the estimated cost to reach the goal (h(n)). The node with the lowest f-score is prioritized for evaluation.

- A* Algorithm is one of the best and popular techniques used for path finding and graph traversals.
- A lot of games and web-based maps use this algorithm for finding the shortest path efficiently
- It is essentially a best first search algorithm.

Working-

* It maintains a tree of paths originating at the start node. It extends those paths one edge at a time. It continues until its termination criterion is satisfied.

* A* Algorithm extends the path that minimizes the following function-

$$\mathbf{f(n)=g(n)+h(n)}$$

* 'n' is the last node on the path

- g(n) is the cost of the path from start node to node 'n'
- h(n) is a heuristic function that estimates cost of the cheapest path from node 'n' to the goal node

Algorithm-

* The implementation of A* Algorithm involves maintaining two lists- OPEN and CLOSED.

* OPEN contains those nodes that have been evaluated by the heuristic function but have not been expanded into successors yet.

* CLOSED contains those nodes that have already been visited.

Step-01:

Define a list OPEN.

Initially, OPEN consists solely of a single node, the start node S.

Step-02

If the list is empty, return failure and exit.

Step-03:

Remove node n with the smallest value of f(n) from OPEN and move it to list CLOSED. If node n is a goal state, return success and exit

Step-04:

Expand node n.

Step-05:

If any successor to n is the goal node, return success and the solution by tracing the path from goal node to 5. Otherwise, go to Step-06.

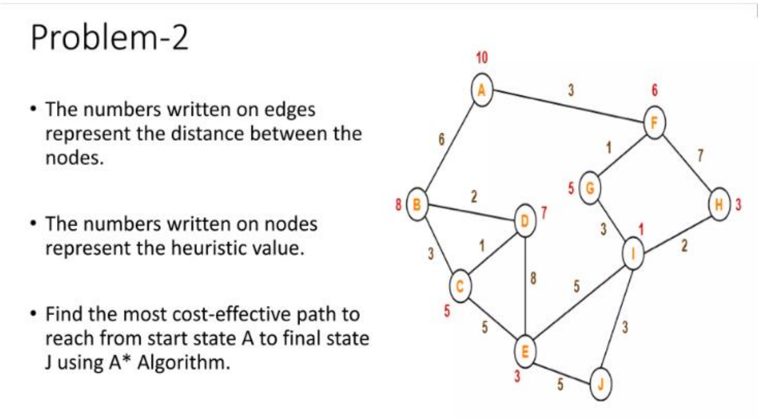
Step-06:

For each successor node, Apply the evaluation function f to the node. If the node has not been in either list, add it to OPEN.

Step-07:

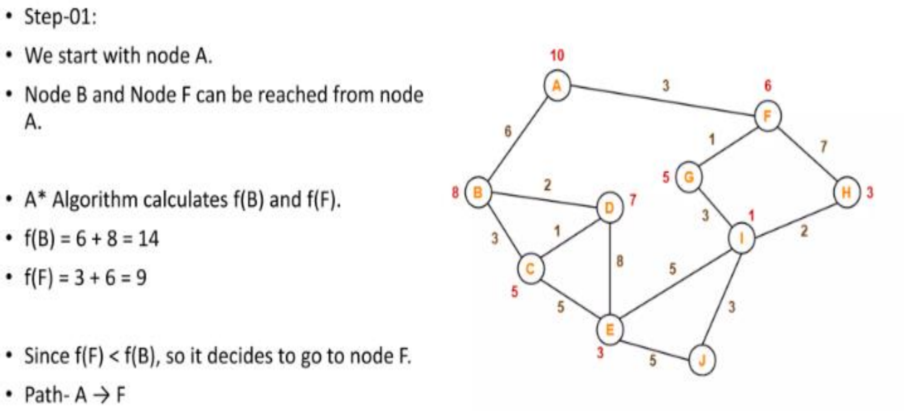
Go back to Step-02.

EXAMPLE



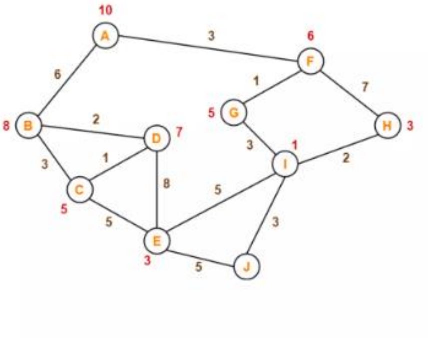
Problem-2

- The numbers written on edges represent the distance between the nodes.
- The numbers written on nodes represent the heuristic value.
- Find the most cost-effective path to reach from start state A to final state J using A* Algorithm.

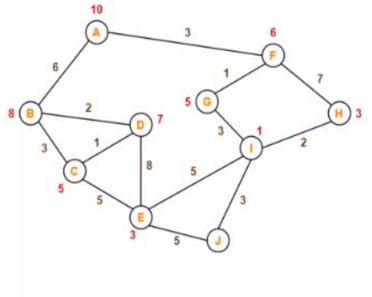


- Step-01:
- We start with node A.
- Node B and Node F can be reached from node A.
- A* Algorithm calculates f(B) and f(F).
- f(B) = 6 + 8 = 14
- f(F) = 3 + 6 = 9
- Since f(F) < f(B), so it decides to go to node F.
- Path- A → F

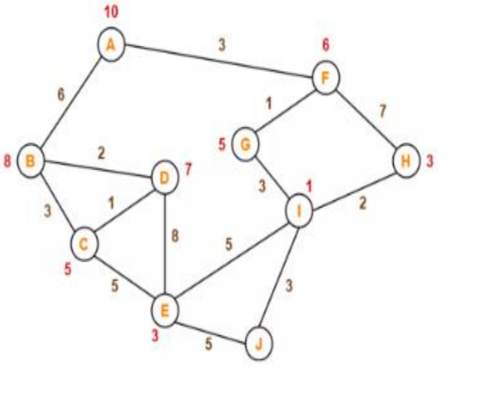
- Step-02:
- Node G and Node H can be reached from node F.
- A* Algorithm calculates f(G) and f(H).
- f(G) = (3+1) + 5 = 9
- f(H) = (3+7) + 3 = 13
- Since f(G) < f(H), so it decides to go to node G.
- Path- A → F → G



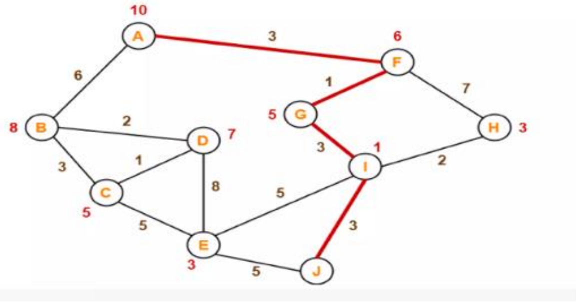
- Step-03:
- Node I can be reached from node G.
- A* Algorithm calculates f(I).
- f(I) = (3+1+3) + 1 = 8
- It decides to go to node I.
- Path- A → F → G → I



- Step-04:
- Node E, Node H and Node J can be reached from node I.
- A* Algorithm calculates f(E), f(H) and f(J).
- f(E) = (3+1+3+5) + 3 = 15
- f(H) = (3+1+3+2) + 3 = 12
- f(J) = (3+1+3+3) + 0 = 10
- Since f(J) is least, so it decides to go to node J.
- Path- A → F → G → I → J

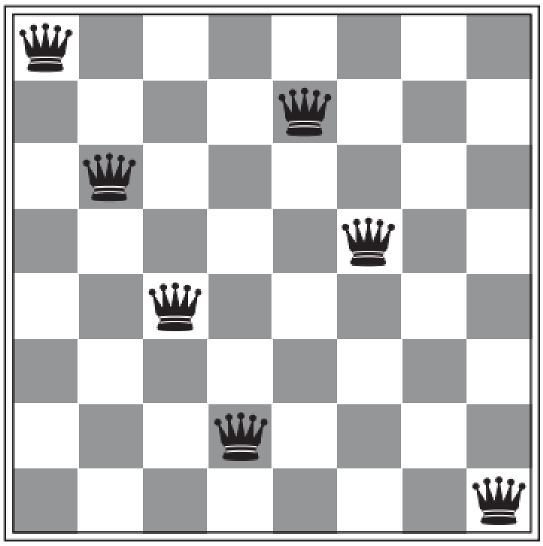


- This is the required shortest path from node A to node J.
-



8-QUEENS PROBLEM:

The goal of the 8-queens problem is to place eight queens on a chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal.) Below Figure shows an attempted solution that fails: the queen in the rightmost column is attacked by the queen at the top left.



Almost a solution to the 8-queens problem.

Although efficient special-purpose algorithms exist for this problem and for the whole n-queens family, it remains a useful test problem for search algorithms. There are two main kinds of formulation. An incremental formulation involves operators that augment the state description, starting with an empty state; for the 8-queens problem, this means that each action adds a queen to the state. A complete-state formulation starts with all 8 queens on the board and moves them around. In either case, the path cost is of no interest because only the final state counts. The first incremental formulation one might try is the following

- **States**: Any arrangement of 0 to 8 queens on the board is a state.

- **Initial state**: No queens on the board.

- **Actions**: Add a queen to any empty square.

- **Transition model**: Returns the board with a queen added to the specified square.

- **Goal test**: 8 queens are on the board, none attacked.

In this formulation, we have 64 · 63 ··· 57 ≈ 1.8 × 1014 possible sequences to investigate. A better formulation would prohibit placing a queen in any square that is already attacked

- States: All possible arrangements of n queens ($0 \le n \le 8$), one per column in the leftmost n columns, with no queen attacking another.

- Actions: Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen

This formulation reduces 8 queens state space from 1.8×10^{14} to just 2057 and solutions are easy to find.

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

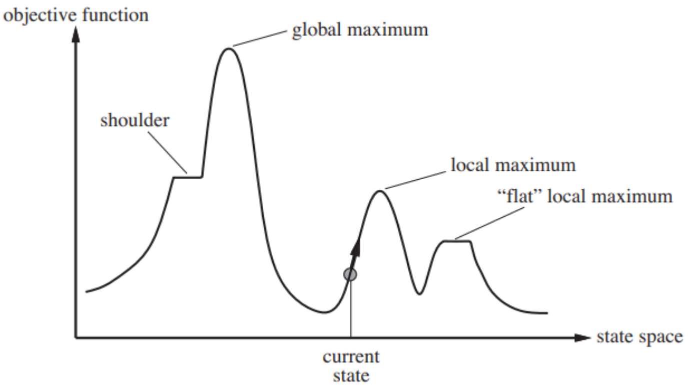
These algorithms are suitable for problems in which all that matters is the solution state, not the path cost to reach it. The family of local search algorithms includes methods inspired by statistical physics (simulated annealing) and evolutionary biology (genetic algorithms).

The search algorithms that we have seen so far are designed to explore search spaces systematically. This systematicity is achieved by keeping one or more paths in memory and by recording which alternatives have been explored at each point along the path. When a goal is found, the path to that goal also constitutes a solution to the problem. In many problems, however, the path to the goal is irrelevant. For example, in the 8-queens problem what matters is the final configuration of queens, not the order in which they are added. The same general property holds for many important applications such as integrated-circuit design, factory-floor layout, job-shop scheduling, automatic programming, telecommunications network optimization, vehicle routing, and portfolio management.

If the path to the goal does not matter, we might consider a different class of algorithms, ones that do not worry about paths at all. Local search algorithms operate using a single current node (rather than multiple paths) and generally move only to neighbours of that node. Typically, the paths followed by the search are not retained. Although local search algorithms are not systematic, they have two key advantages: (1) they use very little memory—usually a constant amount; and (2) they can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable.

In addition to finding goals, local search algorithms are useful for solving pure optimization problems, in which the aim is to find the best state according to an objective

To understand local search, we find it useful to consider the state-space landscape (as Shown in below figure), A landscape has both “location” (defined by the state) and “elevation” (defined by the value of the heuristic cost function or objective function). If elevation corresponds to cost, then the aim is to find the lowest valley—a global minimum; if elevation corresponds to an objective function, then the aim is to find the highest peak—a global maximum. (You can convert from one to the other just by inserting a minus sign.) Local search algorithms explore this landscape. A complete local search algorithm always finds a goal if one exists; an optimal algorithm always finds a global minimum/maximum.



A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

Hill Climbing Algorithm in Artificial Intelligence

Hill-climbing search:

Hill climbing is a simple optimization algorithm used in Artificial Intelligence (AI) to find the best possible solution for a given problem. It belongs to the family of local search algorithms and is often used in optimization problems where the goal is to find the best solution from a set of possible solutions.

- In Hill Climbing, the algorithm starts with an initial solution and then iteratively makes small changes to it in order to improve the solution. These changes are based on a heuristic function that evaluates the quality of the solution. The algorithm continues to make these small changes until it reaches a local maximum, meaning that no further improvement can be made with the current set of moves.

- There are several variations of Hill Climbing, including steepest ascent Hill Climbing, first-choice Hill Climbing, and simulated annealing. In steepest ascent Hill Climbing, the algorithm evaluates all the possible moves from the current solution and selects the one that leads to the best improvement. In first-choice Hill Climbing, the algorithm randomly selects a move and accepts it if it leads to an improvement, regardless of whether it is the best move. Simulated annealing is a probabilistic variation of Hill Climbing that allows the algorithm to occasionally accept worse moves in order to avoid getting stuck in local maxima.

Hill Climbing can be useful in a variety of optimization problems, such as scheduling, route planning, and resource allocation.

Types of Hill Climbing

1. **Simple Hill climbing:**

It examines the neighboring nodes one by one and selects the first neighboring node which optimizes the current cost as the next node.

Algorithm for Simple Hill climbing :

- Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make the initial state as the current state.
- Loop until the solution state is found or there are no new operators present which can be applied to the current state.
    - Select a state that has not been yet applied to the current state and apply it to produce a new state.
    - Perform these to evaluate the new state.
        - If the current state is a goal state, then stop and return success.
        - If it is better than the current state, then make it the current state and proceed further.
        - If it is not better than the current state, then continue in the loop until a solution is found.
- Exit from the function.

2. Steepest-Ascent Hill climbing:

It first examines all the neighboring nodes and then selects the node closest to the solution state as of the next node.

Algorithm for Steepest Ascent Hill climbing :

- Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make the initial state as the current state.
- Repeat these steps until a solution is found or the current state does not change
    - Select a state that has not been yet applied to the current state.
    - Initialize a new ‘best state’ equal to the current state and apply it to produce a new state.
    - Perform these to evaluate the new state
        - If the current state is a goal state, then stop and return success.
        - If it is better than the best state, then make it the best state else continue the loop with another new state.
    - Make the best state as the current state and go to Step 2 of the second point.
- Exit from the function.

3. Stochastic hill climbing:

It does not examine all the neighboring nodes before deciding which node to select. It just selects a neighboring node at random and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another.

- Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make the initial state the current state.
- Repeat these steps until a solution is found or the current state does not change.
  - Select a state that has not been yet applied to the current state.
  - Apply the successor function to the current state and generate all the neighbor states.
  - Among the generated neighbor states which are better than the current state choose a state randomly (or based on some probability function).
  - If the chosen state is the goal state, then return success, else make it the current state and repeat step 2 of the second point.

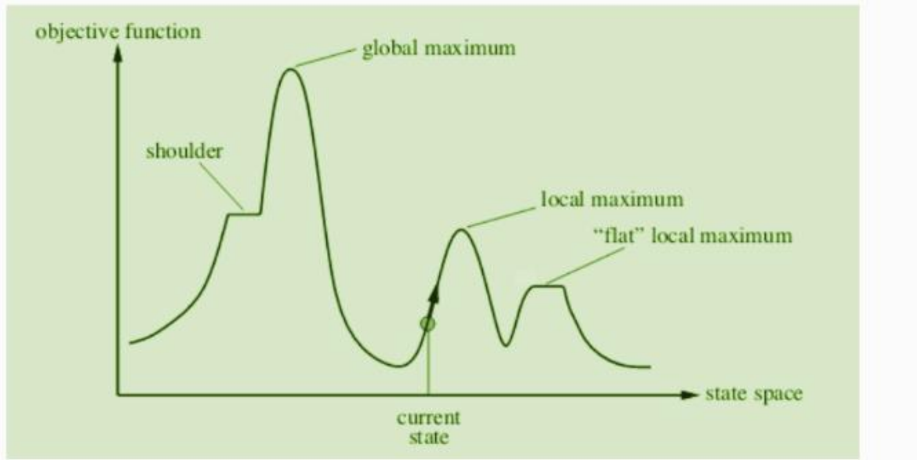
  - Exit from the function.

State Space diagram for Hill Climbing

The state-space diagram is a graphical representation of the set of states our search algorithm can reach vs the value of our objective function(the function which we wish to maximize).

- **X-axis:** denotes the state space ie states or configuration our algorithm may reach.
- **Y-axis:** denotes the values of objective function corresponding to a particular state.

The best solution will be a state space where the objective function has a maximum value(global maximum).



Different regions in the State Space Diagram:

- **Local maximum:** It is a state which is better than its neighboring state however there exists a state which is better than it(global maximum). This state is better because here the value of the objective function is higher than its neighbors.

- **Global maximum:** It is the best possible state in the state space diagram. This is because, at this stage, the objective function has the highest value.
- **Plateau/flat local maximum:** It is a flat region of state space where neighboring states have the same value.
- **Ridge:** It is a region that is higher than its neighbors but itself has a slope. It is a special kind of local maximum.

- **Current state:** The region of the state space diagram where we are currently present during the search.
- **Shoulder:** It is a plateau that has an uphill edge.

Genetic algorithms

Before understanding the Genetic algorithm, let's first understand basic terminologies to better understand this algorithm:

- **Population:** Population is the subset of all possible or probable solutions, which can solve the given problem.
- **Chromosomes:** A chromosome is one of the solutions in the population for the given problem, and the collection of gene generate a chromosome.
- **Gene:** A chromosome is divided into a different gene, or it is an element of the chromosome.
- **Allele:** Allele is the value provided to the gene within a particular chromosome.
- **Fitness Function:** The fitness function is used to determine the individual's fitness level in the population. It means the ability of an individual to compete with other individuals. In every iteration, individuals are evaluated based on their fitness function.
- **Genetic Operators:** In a genetic algorithm, the best individual mate to regenerate offspring better than parents. Here genetic operators play a role in changing the genetic composition of the next generation.
- **Selection**

After calculating the fitness of every existent in the population, a selection process is used to determine which of the individualities in the population will get to reproduce and produce the seed that will form the coming generation.

Types of selection styles available

- **Roulette wheel selection**
- **Event selection**
- **Rank- grounded selection**

How Genetic Algorithm Work?

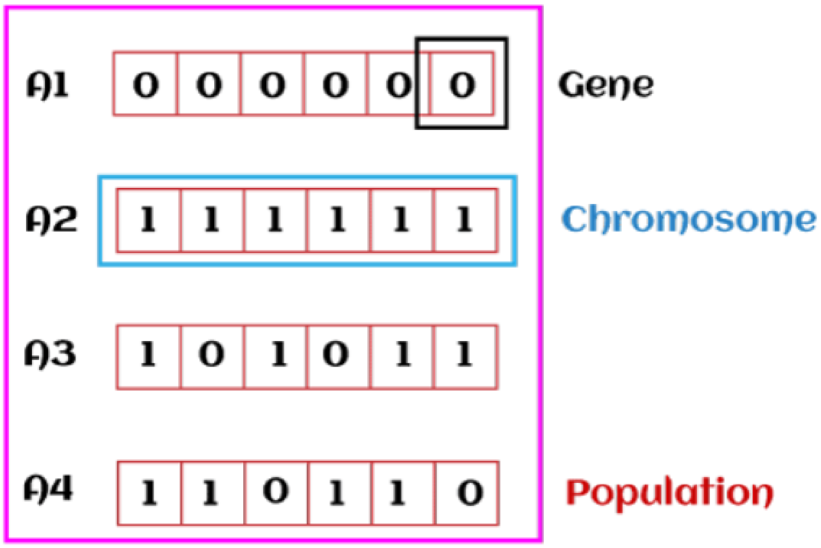
The genetic algorithm works on the evolutionary generational cycle to generate high-quality solutions. These algorithms use different operations that either enhance or replace the population to give an improved fit solution.

It basically involves five phases to solve the complex optimization problems, which are given as below:

- **Initialization**
- **Fitness Assignment**
- **Selection**
- **Reproduction**
- **Termination**

1. Initialization

The process of a genetic algorithm starts by generating the set of individuals, which is called population. Here each individual is the solution for the given problem. An individual contains or is characterized by a set of parameters called Genes. Genes are combined into a string and generate chromosomes, which is the solution to the problem. One of the most popular techniques for initialization is the use of random binary strings.



2. Fitness Assignment

Fitness function is used to determine how fit an individual is? It means the ability of an individual to compete with other individuals. In every iteration, individuals are evaluated based on their fitness function. The fitness function provides a fitness score to each individual. This score further determines the probability of being selected for reproduction. The high the fitness score, the more chances of getting selected for reproduction.

3. Selection

The selection phase involves the selection of individuals for the reproduction of offspring. All the selected individuals are then arranged in a pair of two to increase reproduction. Then these individuals transfer their genes to the next generation.

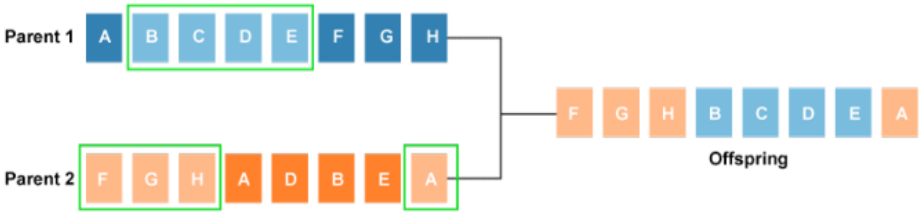
There are three types of Selection methods available, which are:

- Roulette wheel selection
- Tournament selection
- Rank-based selection

4. Reproduction

After the selection process, the creation of a child occurs in the reproduction step. In this step, the genetic algorithm uses two variation operators that are applied to the parent population. The two operators involved in the reproduction phase are given below:

Crossover: The crossover plays a most significant role in the reproduction phase of the genetic algorithm. In this process, a crossover point is selected at random within the genes. Then the crossover operator swaps genetic information of two parents from the current generation to produce a new individual representing the offspring.

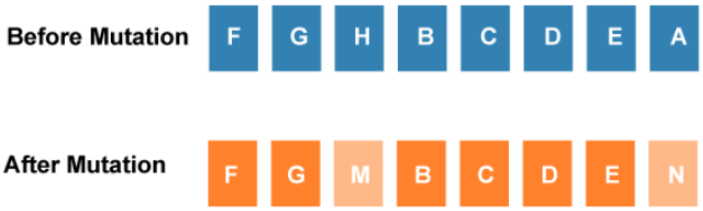


The genes of parents are exchanged among themselves until the crossover point is met. These newly generated offspring are added to the population. This process is also called or crossover. Types of crossover styles available:

- One point crossover
- Two-point crossover
- Livery crossover
- Inheritable Algorithms crossover

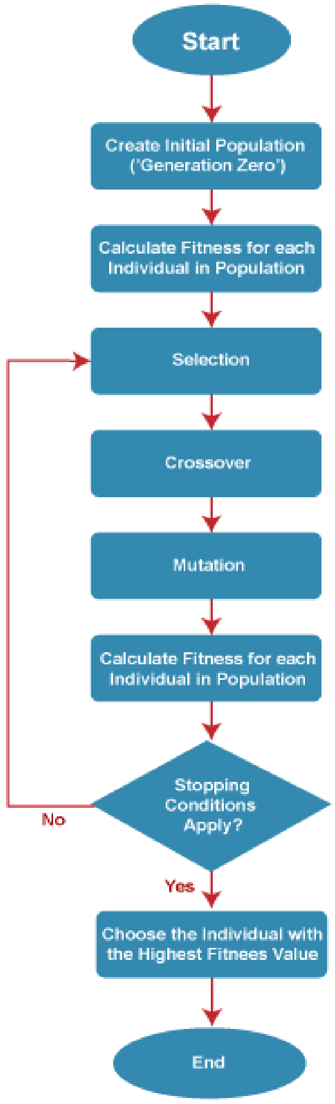
Mutation
The mutation operator inserts random genes in the offspring (new child) to maintain the diversity in the population. It can be done by flipping some bits in the chromosomes.
Mutation helps in solving the issue of premature convergence and enhances diversification. The below image shows the mutation process:
Types of mutation styles available,

- **Flip bit mutation**
- **Gaussian mutation**
- **Exchange/Swap mutation**



Termination

After the reproduction phase, a stopping criterion is applied as a base for termination. The algorithm terminates after the threshold fitness solution is reached. It will identify the final solution as the best solution in the population.


Start
Create Initial Population ('Generation Zero')
Calculate Fitness for each Individual in Population
Selection
Crossover
Mutation
Calculate Fitness for each Individual in Population
Stopping Conditions Apply?
No
Yes
Choose the Individual with the Highest Fitnees Value
End

Advantages of Genetic Algorithm

- The parallel capabilities of genetic algorithms are best.
- It helps in optimizing various problems such as discrete functions, multi-objective problems, and continuous functions.

Limitations of Genetic Algorithms

- Genetic algorithms are not efficient algorithms for solving simple problems.
- It does not guarantee the quality of the final solution to a problem.
- Repetitive calculation of fitness values may generate some computational challenges.