Blind/uninformed search

- Depth first search
- Breadth first search
- These search algorithms explore the state space in the same order, irrespective of the goal to be achieved
  - That's why called as blind or uninformed search
- These search algorithms maintain a list, called OPEN, of candidate nodes.
- If the algorithms operates OPEN as a stack, the behaviour is depth first
- If the algorithms operates OPEN as a quest, the beahavior is breadth first
- We would like the algorithm could make a guess as to which of the candidates is more likely to lead to goal
  - It would have a chance of finding the goal node faster.

Informed search

- one that uses problem-specific knowledge beyond the definition of the problem itself
- 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.

Heuristic Search Techniques

- DFS and BFS may require too much memory to generate an entire state space
  - In these cases heuristic search is used.
- Heuristics help us to reduce the size of the search space.
- An evaluation function is applied to each state to assess how promising it is in leading to the goal.

Heuristic Search Techniques

- Heuristic searches incorporate the use of domain specific knowledge in the process of choosing which node to visit next in the search process.
- This domain specific knowledge is used in form of heuristics.
- Using the domain knowledge is meant to speed up the search process.
- Search methods that use heuristics are described as weak methods.
  - Since the knowledge used is weak in that it usually helps but does not always help to find a solution.
    - Heuristic functions are not guaranteed to be completely accurate.

Calculating Heuristics

- Heuristic values are greater than and equal to zero for all nodes.
- Heuristic values are seen as an approximate cost of finding a solution.
- A heuristic values of 0 indicates that the state is a goal state.
- A heuristic that never overestimates the cost to the goal is referred to as an admissible heuristic.
- Not all heuristics are necessarily admissible.
- A heuristic value of infinity indicates that the state is a dead end and is not going to lead anywhere.

Calculating Heuristics

- A good heuristic must not take long to compute
- Heuristics are often defined on a simplified or relaxed version of the problem e.g.
  - The number of tiles that are out of place
- A heuristic function h1 is better than some heuristic function h2 if fewer nodes are expanded during the search when h1 is used than when h2 is used.

Calculating Heuristics

- Each of the following could serve as a heuristic function for the 8-puzzle problem
  - Number of tiles out of place
    - Count the number of tiles out of place in each state compared to the goal.
  - Sum all the distance that the tiles are out of place.
- Experience has shown that it is difficult to devise heuristic functions.
- Furthermore, heuristics are fallible and are by no means perfect.

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.

Heuristic search algorithms

- Examples
  - Best First Search
    - Greedy Best First Search
    - A* Best First Search
  - Hill Climbing

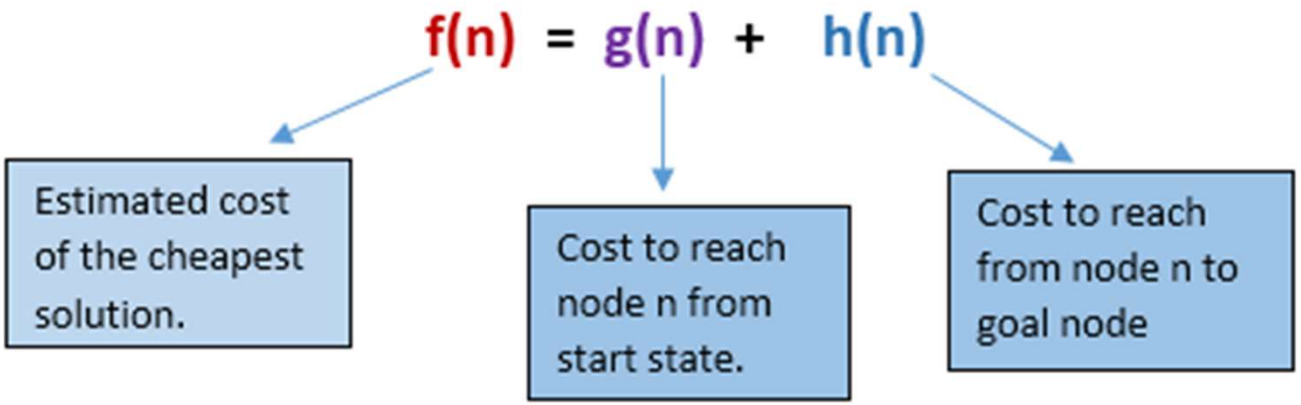
Best-first search

- Best-first search allows us to take the advantages of both algorithms
- It is the combination of depth-first search and breadth-first search algorithms.
- It uses the heuristic function and search.
- BFS uses the concept of a Priority queue and heuristic search.
- This algorithm will traverse the shortest path first in the queue.
- The time complexity of the algorithm is given by O(n*logn).
- Search algorithms that are guaranteed to find the shortest path are called admissible algorithms.
  - The Best first search is an example of an admissible algorithm

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.

- It would be more efficient to implement OPEN as a priority queue


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

Greedy vs. A* Best-first Search Algorithm

- The only difference between Greedy BFS and A* BFS is in the evaluation function.
- For Greedy BFS the evaluation function is f(n) = h(n)
  - It is possible that Greedy BFS can choose a longer solution in terms of number of moves required to reach goal node.
    - Because algorithm is not taking into account the cost of reaching the states.
- while for A* the evaluation function is f(n) = g(n) + h(n).

A* search

- 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.
- A* search is the most commonly known form of best-first search.
- It has combined features of Uniform cost search (UCS) and greedy best-first search, by which it solve the problem efficiently.
- This search algorithm expands less search tree and provides optimal result faster.
- A* is more optimal of the two approaches as it also takes into consideration the total distance travelled so far i.e. g(n).

Comparison

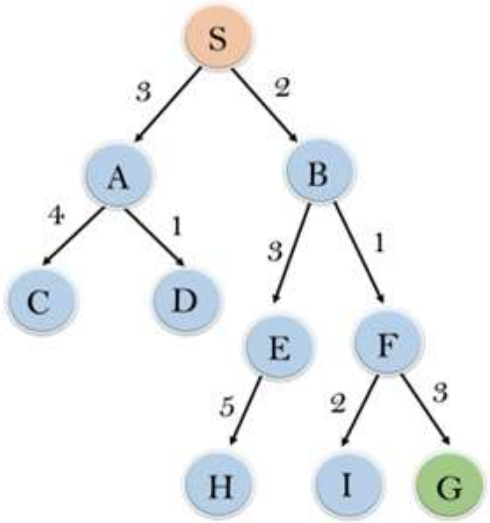
- Both Greedy BFS and A* are the Best first searches
- but Greedy BFS is neither complete nor optimal
  - It can behave as an unguided depth-first search in the worst case scenario.
  - It can get stuck in a loop as DFS.
- whereas A* is both complete and optimal
  - A* algorithm is able to come up with is a more optimal path than Greedy BFS
  - However, A* uses more memory than Greedy BFS, but it guarantees that the path found is optimal.
- Greedy Best-First-Search is *not* guaranteed to find a shortest path.

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.

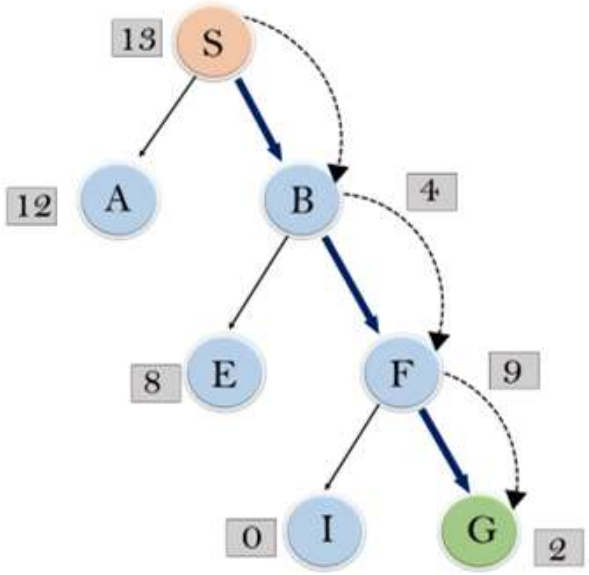
Example

- 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**
- **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 maximum number of nodes that are created.
  - The worst case time complexity of Greedy best first search is $O(b^m)$.
- **Space Complexity:**
  - The maximum number of nodes that are stored in memory.
  - 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.
- **Depth** – Length of the shortest path from initial state to goal state.
- **Branching Factor** – The average number of child nodes in the problem space graph.

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.
- A* becomes impractical when the search space is huge.
- However, A* also guarantees that the found path between the starting node and the goal node is the optimal one and that the algorithm eventually terminates. Greedy BFS, on the other hand, uses less memory, but does not provide the optimality and completeness guarantees of A*. So, which algorithm is the "best" depends on the context, but both are "best"-first searches.

PRACTICE PROBLEMS BASED ON A* ALGORITHM

- **Problem-01**
- Given an initial state of a 8-puzzle problem and final state to be reached-

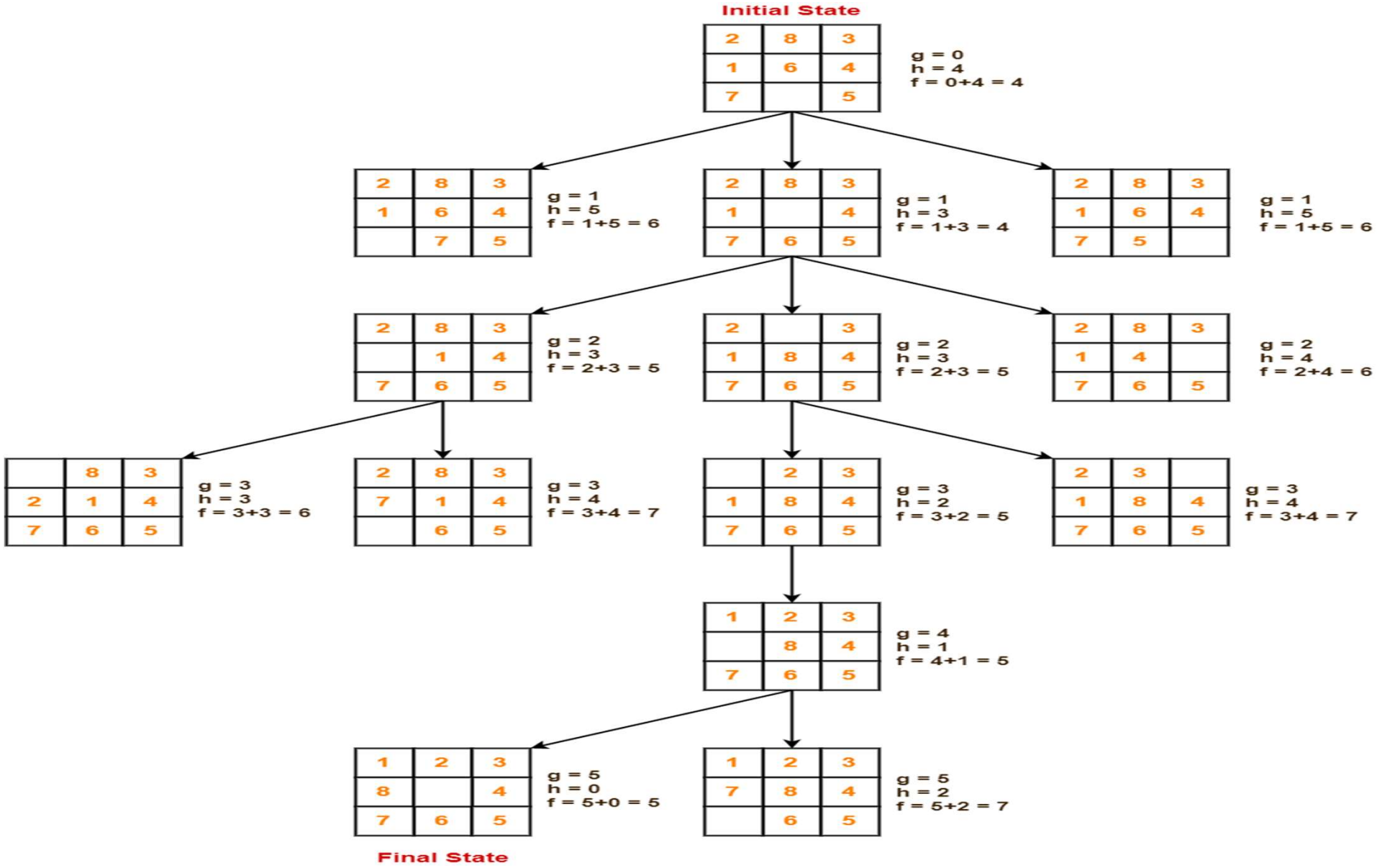
Initial State

Final State

- Find the most cost-effective path to reach the final state from initial state using A* Algorithm.
- Consider g(n) = Depth of node and h(n) = Number of misplaced tiles.

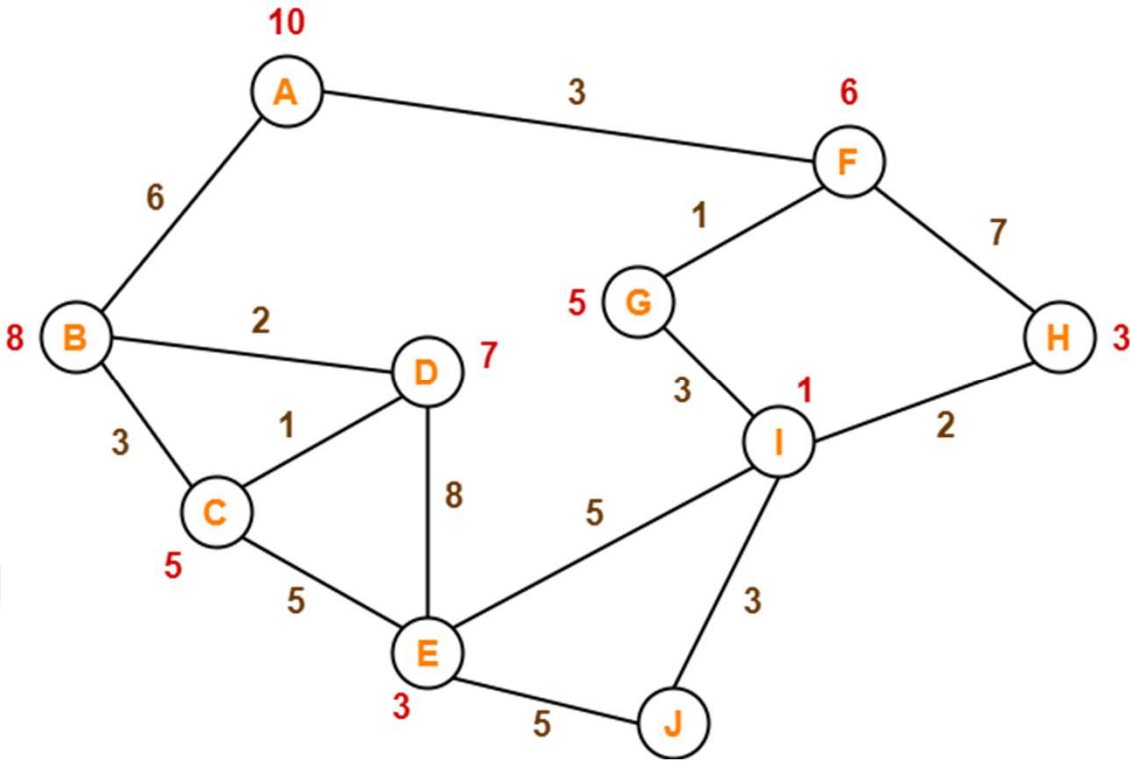
Solution

- A* Algorithm maintains a tree of paths originating at the initial state.
- It extends those paths one edge at a time.
- It continues until final state is reached.


Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 3
1 8 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 8 3
1 4
7 6 5
g = 2
h = 4
f = 2+4 = 6
8 3
2 1 4
7 6 5
g = 3
h = 3
f = 3+3 = 6
2 8 3
7 1 4
6 5
g = 3
h = 4
f = 3+4 = 7
2 3
1 8 4
7 6 5
g = 3
h = 2
f = 3+2 = 5
2 3
1 8 4
7 6 5
g = 3
h = 4
f = 3+4 = 7
1 2 3
8 4
7 6 5
g = 4
h = 1
f = 4+1 = 5
1 2 3
8 4
7 6 5
g = 5
h = 0
f = 5+0 = 5
1 2 3
7 8 4
6 5
g = 5
h = 2
f = 5+2 = 7
Final State

Problem 2

- Consider the following graph
- 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.



Solution

- **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**

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)
- A* Algorithm is one of the best path finding algorithms.
- But it does not produce the shortest path always.
  - This is because it heavily depends on heuristics.

Completeness

- Breadth first search and Depth first search are complete for finite state spaces.
- if there is no solution and the state space is infinite then both algorithms (Breadth first search and Depth first search) and will not terminate.
- Greedy Best first search is complete at least for finite domains.
- For infinite state spaces, the completeness property will depend upon the quality of the heuristic function.
- A* algorithm is complete as long as:
  - Branching factor is finite.
  - Cost at every action is fixed.

Quality of solution (Optimality)

- Since greedy BFS considers only h(n), it may find a sub-optimal (with larger number of moves than moves required to find the optimal solution) solution.
- If the heuristic function h(n) is admissible, then A* tree search will always find the least cost path.

Space Complexity

- space in terms of the maximum number of nodes stored in memory
  - Number of nodes in OPEN list
- OPEN list grows
  - linearly for Depth First Search
  - Exponentially for Breadth First Search
  - Depending on the accuracy of heuristic search In case of Greedy BFS
    - If the heuristic function is accurate, the list will grow linearly
    - Otherwise it grows exponentially

Space Complexity

- The space complexity of A* search algorithm is $O(b^d)$
- Imagine searching a uniform tree where every state has b successors.
- The root of the search tree generates b nodes at the first level, each of which generates b more nodes, for a total of b^2 at the second level.
- Each of these generates b more nodes, yielding b^3 nodes at the third level, and so on.
- Now suppose that the solution is at depth d.
- In the worst case, it is the last node generated at that level.
- Then the total number of nodes generated is
  - $b + b^2 + b^3 + \cdots + b^d = O(b^d)$.

Time Complexity

- Time is often measured in terms of the number of nodes generated during the search,
- If goal node exists, in both cases (Breadth first search and Depth first search) time taken to find it depends on where the goal node is.
- The time complexity of greedy and A* search algorithm depends on heuristic function
  - With a perfect heuristic function the search will find the goal in linear time.
  - Otherwise the time required too tends to be exponential in practice (as it is difficult to find a very good 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.

To check accuracy of the heuristic functions

- A perfect heuristic function would only inspect nodes that lead to the goal node.
- Otherwise search would often pick nodes that are not part of the path.
- Effective branching factor
  - No of extra nodes a search algorithm inspects
- Effective branching factor= total nodes expanded/ nodes in the solution
- For a perfect heuristic function effective branching factor would tend to be 1.

Hill Climbing

- Hill Climbing is a heuristic search used for mathematical optimization problems in the field of Artificial Intelligence.
- Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum
- 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

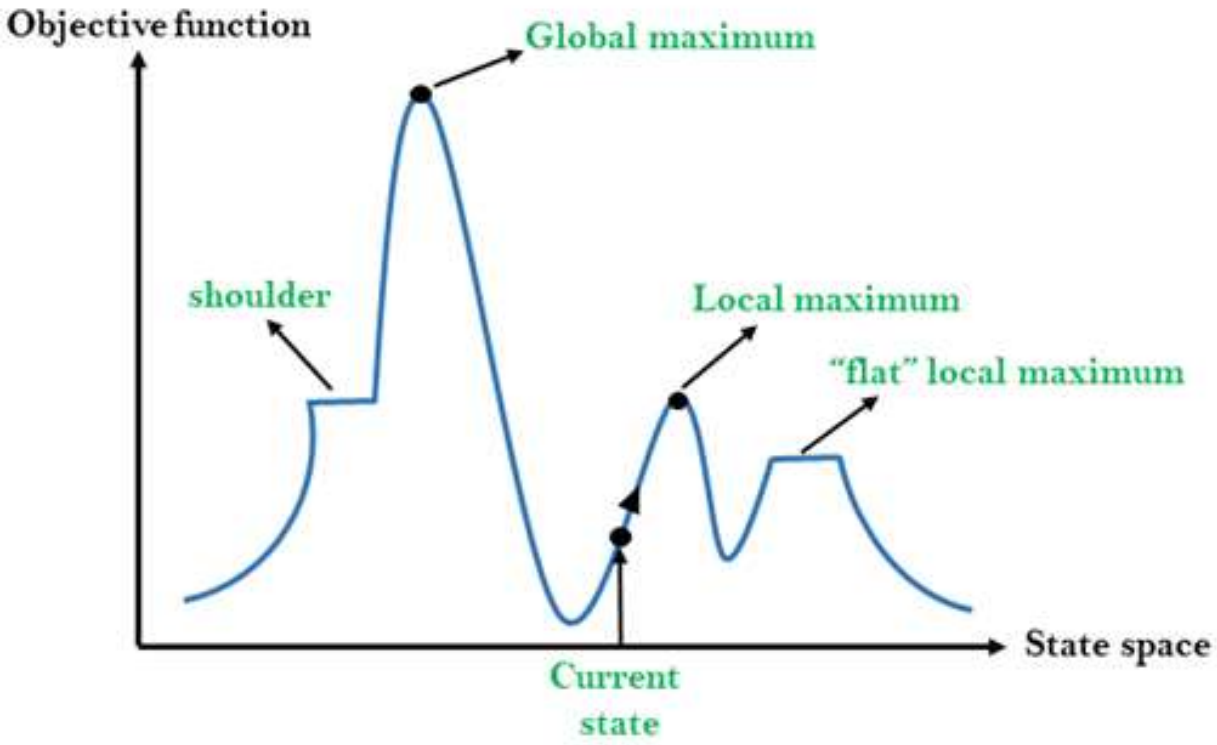
- Following are some main features of Hill Climbing Algorithm:
- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method.
  - *1.Generate possible solutions.*
  - *2. Test to see if this is the expected solution.*
  - *3. If the solution has been found quit else go to step 1.*
- **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.

Properties of Good generators

- Good generators are complete.
  - They eventually generate all possible solutions
- Good generators are nonredundant.
  - They never compromise efficiency by producing the same solution twice.
- Good generators are informed.
  - They use possibility-limiting information, restricting the number of solution they generate.

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


Objective function
Global maximum
shoulder
Local maximum
"flat" local maximum
State space
Current
state

Different regions in the state space landscape

- **Local Maximum:** Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.
- **Global Maximum:** Global maximum is the best possible state of state space landscape. It has the highest value of objective function.
- **Current state:** It is a state in a landscape diagram where an agent is currently present.
- **Flat local maximum:** It is a flat space in the landscape where all the neighbor states of current states have the same value.
- **Shoulder:** It is a plateau region which has an uphill edge.

Types of Hill Climbing Algorithm

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

Simple Hill Climbing

- Simple hill climbing is the simplest way to implement a hill climbing algorithm.
- **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**.
- It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state.
- This algorithm has the following features:
  - Less time consuming
  - Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **a)** Select an operator and apply it to the current state.
- b) Check new state:
  - If it is goal state, then return success and quit.
  - Else if it is better than the current state then assign new state as a current state.
  - Else if not better than the current state, then return to step2.
- **Step 3:** Exit.

Steepest Gradient Ascent hill climbing

- The steepest gradient Ascent algorithm is a variation of simple hill climbing algorithm.
- This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state.
- This algorithm consumes more time as it searches for multiple neighbors
- Once starting candidate solution is decided the algorithm moves up (or down) along the steepest gradient (or descent, if the problem is of minimization) and terminates when the gradient becomes zero.
- The end point of the search is determined by the starting point and the nature of the surface defined by the evaluation function.
  - If the surface is monotonic then the end point is the global optimum
  - Otherwise the search ends up at an optimum that is in some sense closest to the starting point but may only be local optimum.

Algorithm for Steepest-Ascent hill climbing

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
  - *a) Select an operator that has not been yet applied to the current state.*
  - *b) Initialize a new 'best state' equal to current state and apply it to produce a new state.*
  - *c) Perform these to evaluate new state*

    i. If the current state is a goal state, then stop and return success.

    ii. If it is better than best state, then make it best state else continue loop with another new state.
- *d) Make best state as current state and go to Step 2: b) part.*
- **Step 3:** Exit.

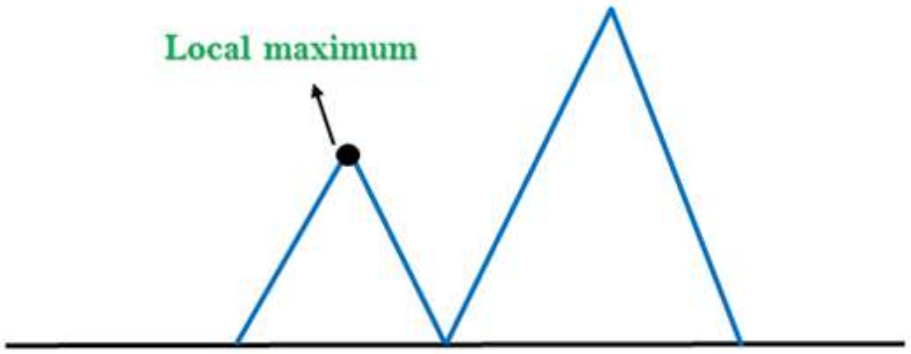
Stochastic hill climbing

- Stochastic hill climbing does not examine for all its neighbor before moving.
- Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

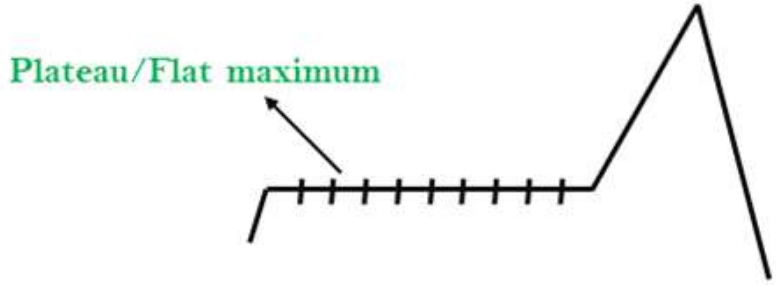
- ***Step 1:*** *Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make* the *initial state* the *current state.*
- ***Step 2:*** *Repeat these steps until a solution is found or* the *current state does not change.*
  - *a) Select an operator that has not been yet applied to the current state.*
  - *b) Apply successor function to the current state and generate all the* neighbor *states.*
  - *c) Among the generated* neighbor *states which are better than* the *current state choose a state randomly (or based on some probability function).*
  - *d) If the* chosen *state is* the *goal state, then return success, else make it* the *current state and repeat step 2: b) part.*
- ***Step 3:*** *Exit.*

Problems in Hill Climbing Algorithm

- **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.



- **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.

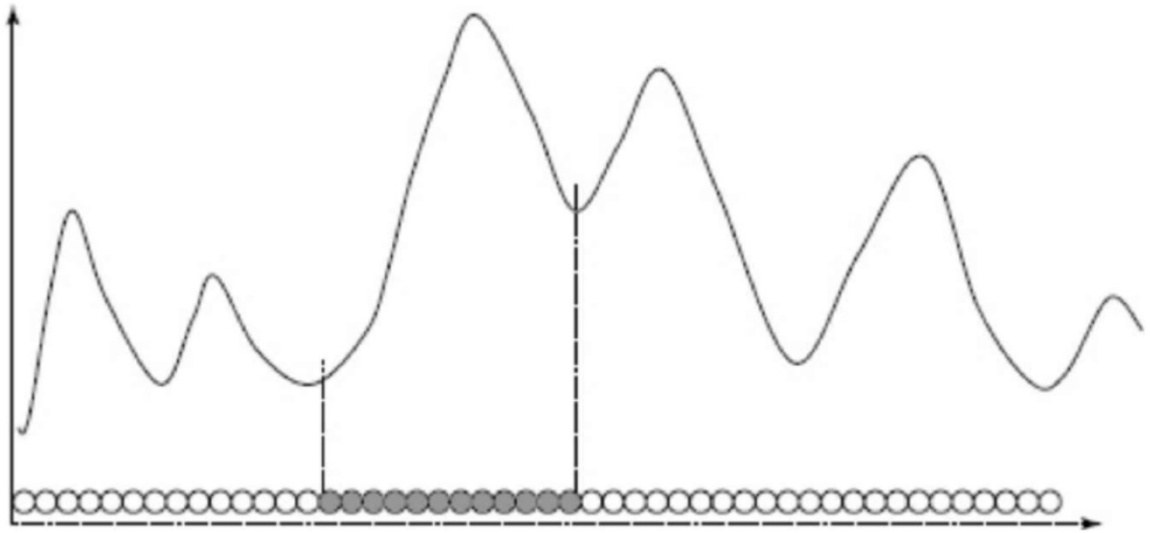


- **Ridges:** A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.
- **Solution:** With the use of bidirectional search, or by moving in different directions, we can improve this problem.
-

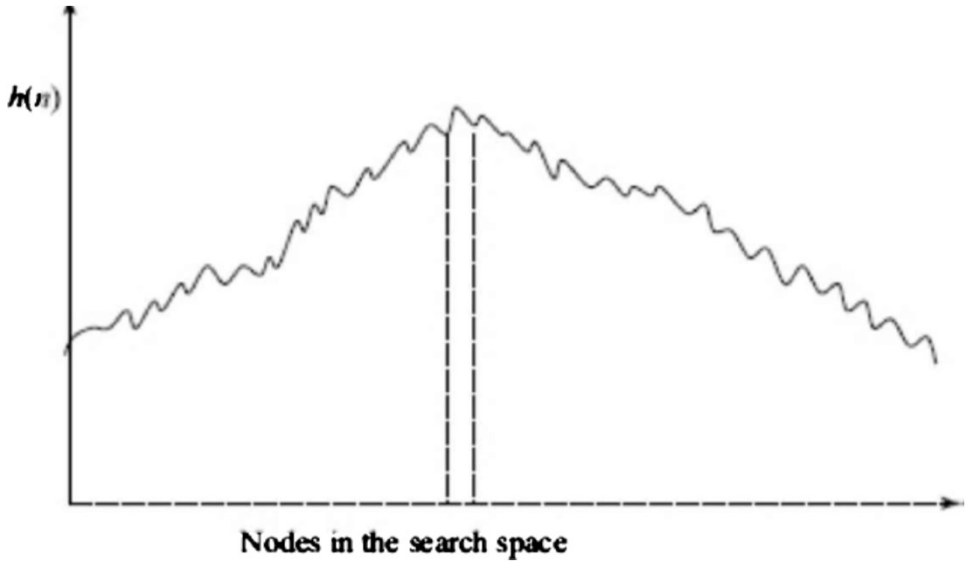
Randomized search algorithms

- To escape local maxima
- Examples
  - Iterated Hill Climbing
    - Does a series of searches from a set of randomly selected different starting points.
    - The hope is that the best value found by at least one of the searches will be global optimum.
  - Simulated Annealing
  - Genetic Algorithms

Shaded nodes will lead to global maximum



- Set of shaded nodes from where the steepest gradient leads to the global maximum.
- Lets call this set the footprint of the hill climbing algorithm
- The likelihood of iterated hill climbing finding the global hill climbing depends upon the size of the footprint.
- If the footprint covers the entire search space then steepest gradient hill climbing itself will work.
- As the footprints gets smaller, a large number of iterations are required to have a good chance of finding the optimum.



- If the footprint is very small then the prohibitively large number of iterations may require and in that case we should look for an alternative approach.

Means-Ends Analysis

- The purpose of means-ends analysis is to identify a procedure that causes a transition from the current state to the goal state or at least to an intermediate state that is closer to the goal state.
- Thus the identified procedure reduces the observed differences between the current state and the goal state.
- The key idea in mean-ends analysis is to reduce differences
- Descriptions of the current state or of the goal state or of the difference between these states may contribute to the identification of difference reducing procedure.

- 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).

How means-ends analysis Works

- Following are the main Steps which describes the working of MEA technique for solving a problem.
  1. First, evaluate the difference between Initial State and goal State.
  2. Select the various operators/procedures which can be applied for each difference.
  3. Apply the operator at each difference, which reduces the difference between the current state and goal state.

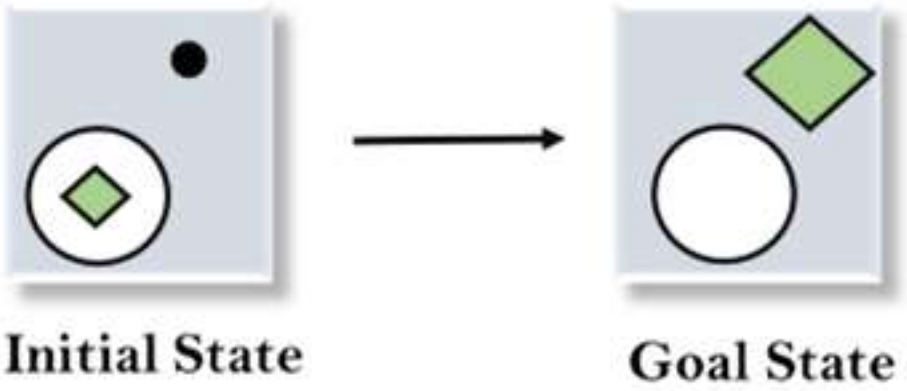
To perform means–ends analysis,

- Until the goal is reached or no more procedures are available,
  - Describe the current state, the goal state, and the difference between the two.
  - Use the difference between the current state and goal state, possibly with the description of the current state or goal state, to select a promising procedure.
  - Use the promising procedure and update the current state.
- If the goal is reached, announce success; otherwise, announce failure.

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/procedure to reduce the differences.
- But sometimes it is possible that an operator/procedure 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**.

- Example of Mean-Ends Analysis



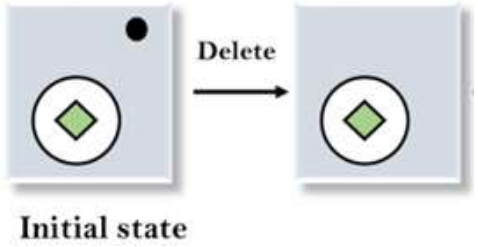
Solution

- 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/procedures.
- The operators/procedures we have for this problem are:
  - **Move**
  - **Delete**
  - **Expand**

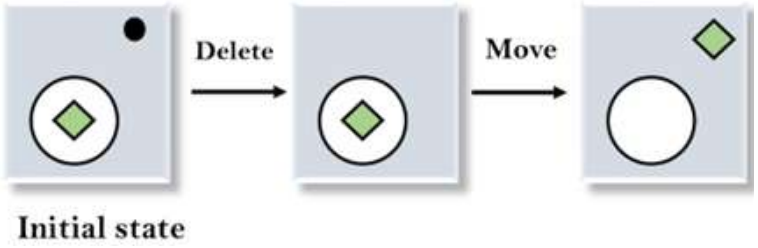
- **1. Evaluating the initial state:** In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.

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.



- **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**.



- **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.