

# Search Algorithms in Artificial Intelligence

SUKANTA GHOSH

# 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.

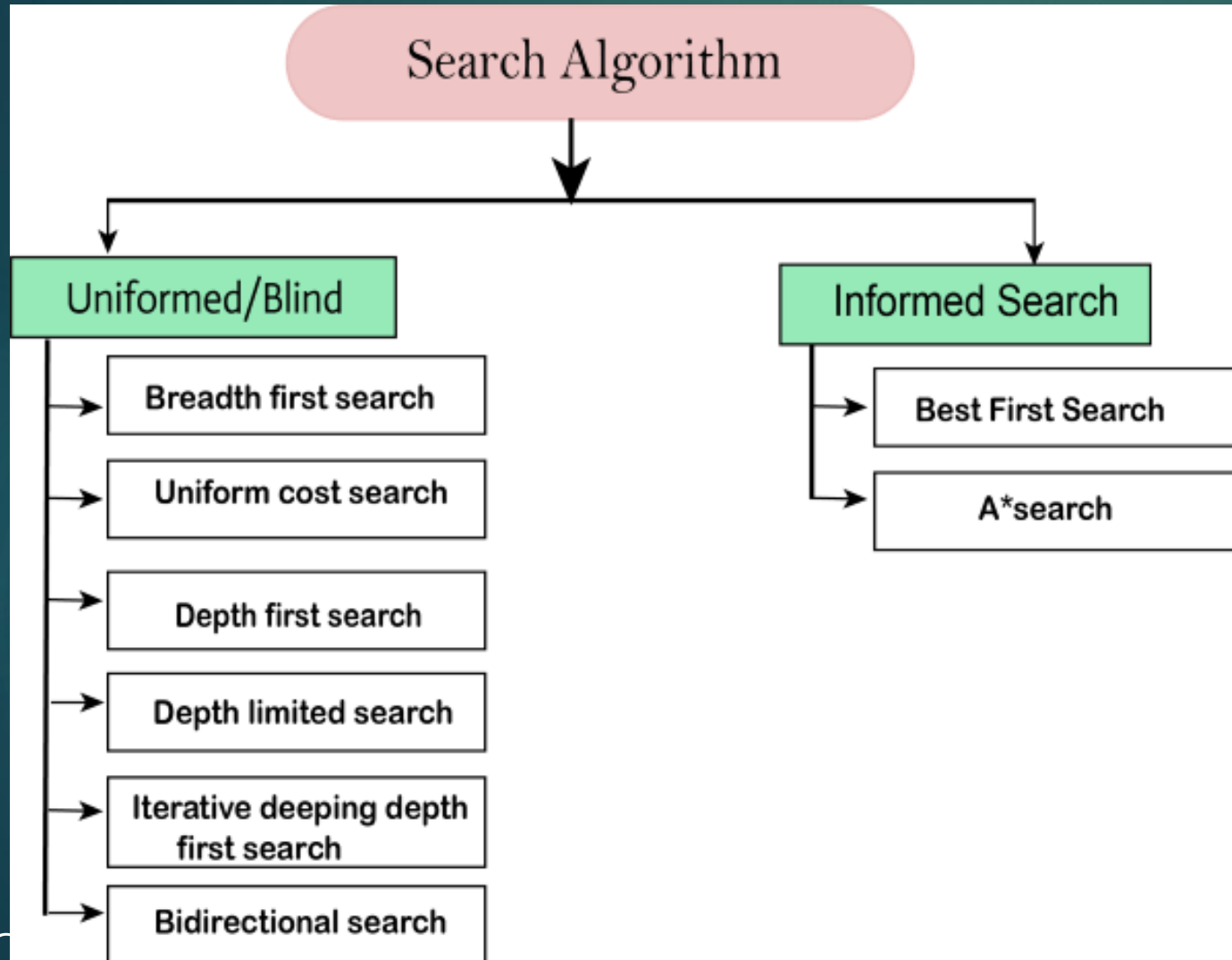
# Search Algorithm Terminologies

- ▶ **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 the possible solutions.

# Properties of Search 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



# Uninformed/Blind Search

- ▶ Does not contain any domain knowledge such as closeness, the location of the goal.
- ▶ 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.
- ▶ No information about the search space, hence known as blind search.

# Informed Search

- ▶ Informed search algorithms use domain knowledge.
- ▶ 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.



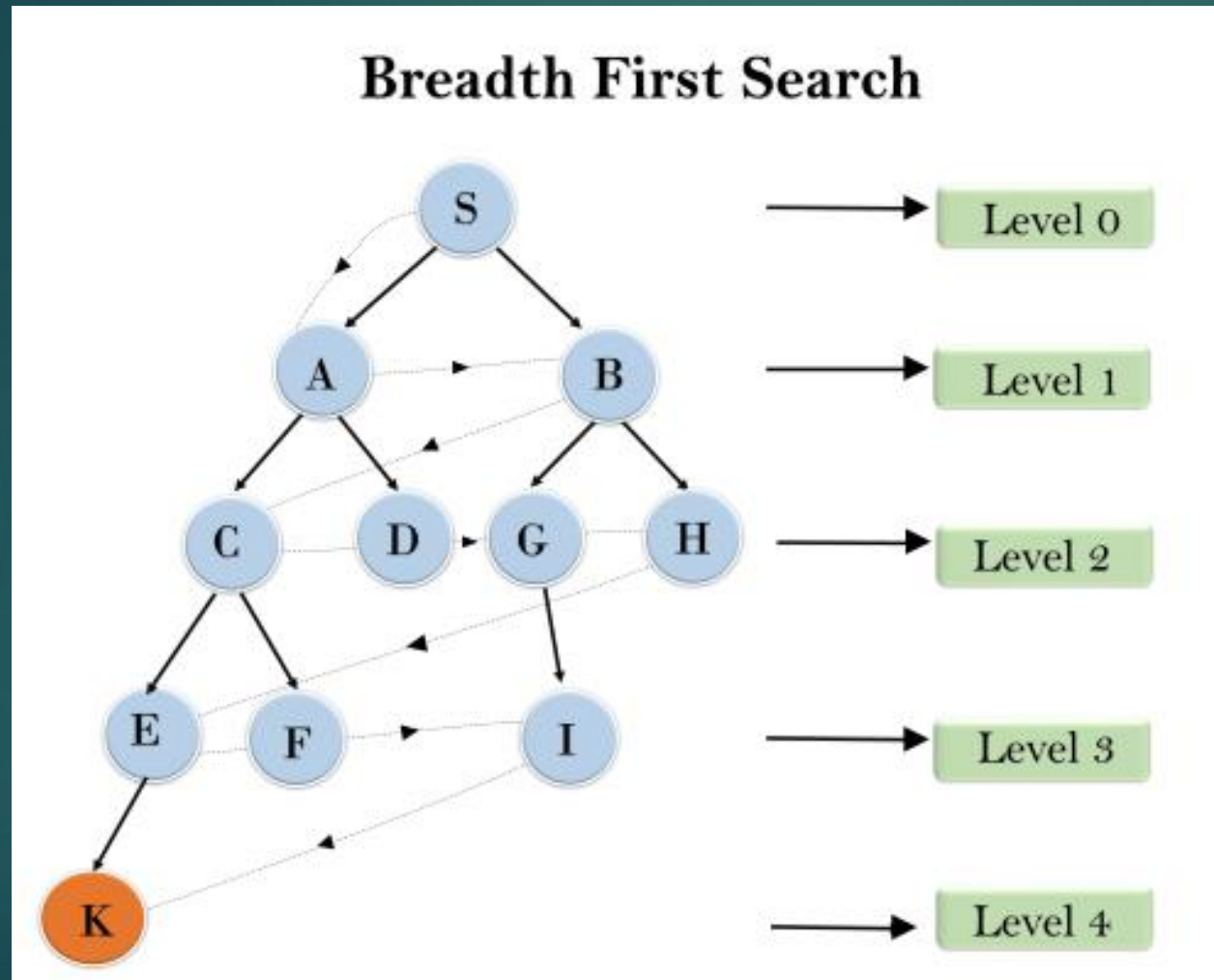
# Uninformed/Blind Search



# Breadth-first Search

- ▶ This algorithm searches breadthwise in a tree or graph.
- ▶ 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.

# Example



# Breadth-first Search

## ► 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.

# Algorithm

**Step 1.** Put the initial node on a list START.

**Step 2.** If (START is empty) or (START=GOAL), terminate search.

**Step 3.** Remove the first node from START, call it node a.

**Step 4.** If (a=GOAL) terminate search with success.

**Step 5.** Else if node a has successor, generate all of them and add them to tail of START.

**Step 6.** Go to step 2

# Algorithm

- Step 1 : Put the initial node on a list START
- Step 2 : If (START is empty) or (START=GOAL) terminate search
- Step 3 : Remove the first node from START. Call this node  $a$
- Step 4 : If ( $a=GOAL$ ) terminate search with success
- Step 5 : Else if node  $a$  has successors, generate all of them and add them at the tail of START
- Step 6 : Goto Step 2.

Fig. 3.3 Algorithm for Breadth-First search

# Breadth-first Search

- ▶ **Time/space 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.
- ▶  $T(b) = 1 + b + b^2 + b^3 + \dots + b^d = O(b^{d+1})$

- ▶ **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.



# Applications

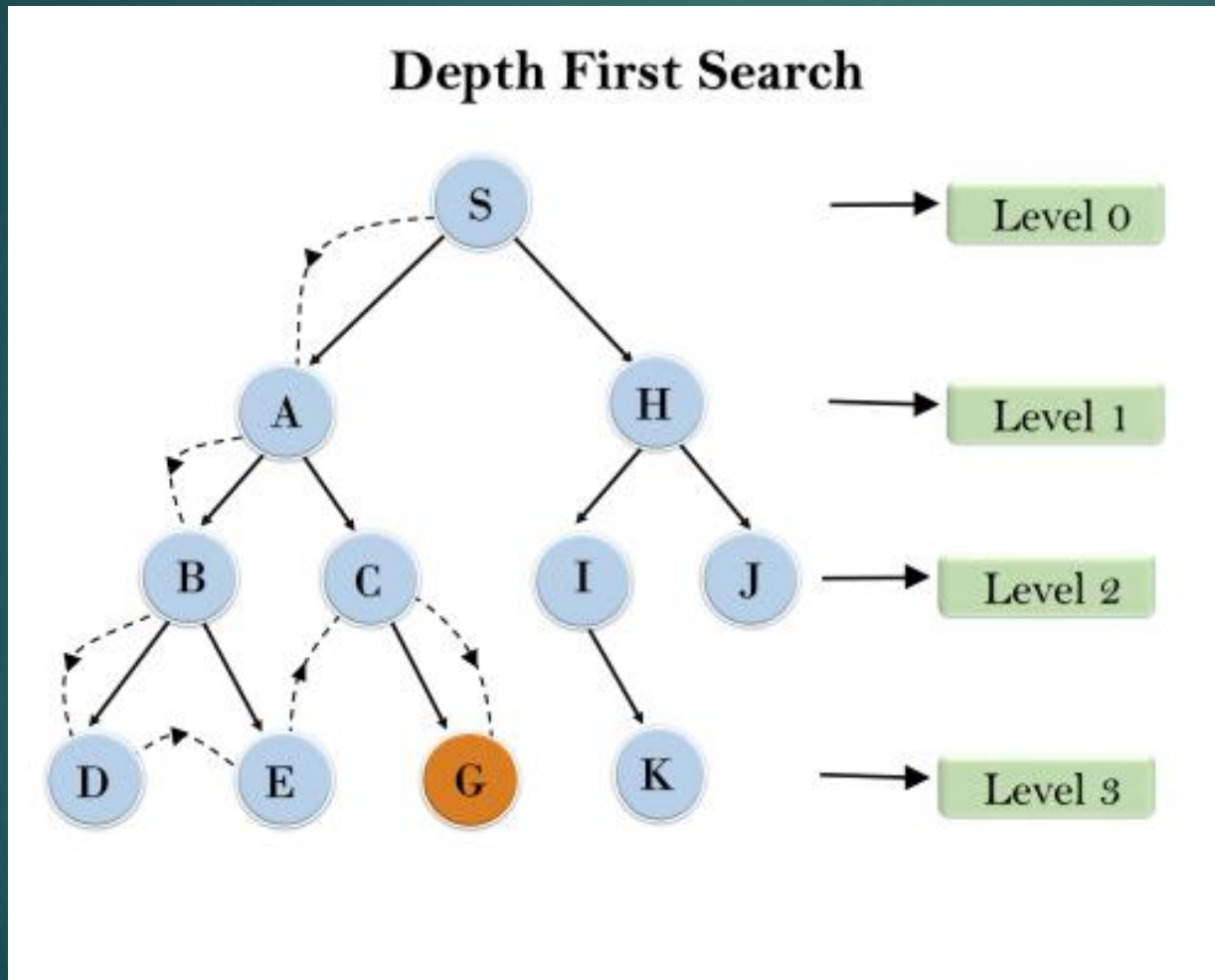
- ▶ Breadth-first search can be utilized to explain many problems in graph theory, for instance:
- ▶ Locating all nodes within one connected component
- ▶ Copying Collection, Cheney's algorithm
- ▶ Locating the shortest path among two nodes  $u$  and  $v$
- ▶ Testing a graph for bipartiteness
- ▶ (Reverse) Cuthill–McKee mesh numbering
- ▶ Ford–Fulkerson method for calculating the maximum flow in a flow network
- ▶ Serialization/Deserialization of a binary tree vs serialization in sorted order, permits the tree to be reconstructed in a competent manner.



# Depth-first Search

- ▶ 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.
- ▶ The process of the DFS algorithm is similar to the BFS algorithm.

# Example



# Depth-first Search

## ► **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.

# Algorithm

**Step 1.** Put the initial node on a list START.

**Step 2.** If (START is empty) or (START=GOAL), terminate search.

**Step 3.** Remove the first node from START, call it node a.

**Step 4.** If (a=GOAL) terminate search with success.

**Step 5.** Else if node a has successor, generate all of them and add them to beginning of START.

**Step 6.** Go to step 2

# Algorithm

Step 1 : Put the initial node on a list START

Step 2 : If (START is empty) or (START=GOAL) terminate search

Step 3 : Remove the first node from START. Call this node  $a$

Step 4 : If ( $a=GOAL$ ) terminate search with success

Step 5 : Else if node  $a$  has successors, generate all of them and add them at the beginning of START

Step 6 : Goto Step 2.

Fig. 3.1 Algorithm for Depth-First search

# Depth-first Search

- ▶ **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 is given by:
- ▶  $T(n) = 1 + n^2 + n^3 + \dots + n^m = O(n^m)$
- ▶ 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.



# Problem with uninformed/blind search

- ▶ Do not have any domain specific knowledge.
- ▶ Process of searching is drastically reduced and inefficient.



# Informed / Heuristics Search

# Informed / Heuristics Search

- ▶ 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.
- ▶ It is also called Heuristic search.
- ▶ Informed search algorithm is more useful for large search space.

# Informed / Heuristics Search

- ▶ Two categories of problem are uses heuristics:
- ▶ Problem for which no exact algorithm are known, and one need to find an approximate and satisfying solution. E.g. Computer vision or speech recognition.
- ▶ Problem for which exact solution are known, but computationally infeasible. E.g. Rubric cube or Chess.

# 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 does not always give the best solution, but it guaranteed to find a good solution in reasonable time.
- ▶ 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.

# Heuristics function

- ▶ Admissibility of the heuristic function is given as:

$$h(n) \leq h^*(n)$$

- ▶ 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.
- ▶ Some simple heuristic functions:
  - ▶ 8-tile puzzle: hamming distance is used
  - ▶ Chess Problem: Material Advantage is used

# Best-first Search Algorithm

- ▶ 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.
- ▶ With the help of best-first search, at each step, we can choose the most promising node.
- ▶ In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

$$f(n) = g(n).$$

# Best first search algorithm

**Step 1.** Put the initial node on a list START.

**Step 2.** If (START is empty) or (START=GOAL), terminate search.

**Step 3.** Remove the first node from START, call it node a.

**Step 4.** If (a=GOAL) terminate search with success.

**Step 5.** Else if node a has successor, generate all of them. Find out how far they are from the goal node. Sort all the children generated so far by the remaining distance from the goal.

**Step 6.** Name the list as START 1 and replace it with list START.

**Step 7.** Go to step 2.



# Algorithm

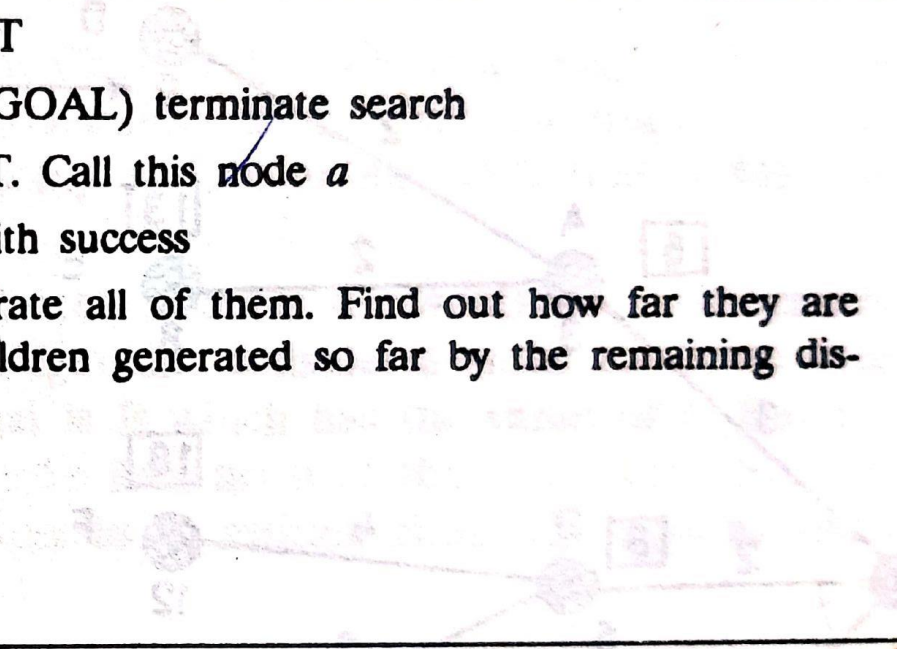
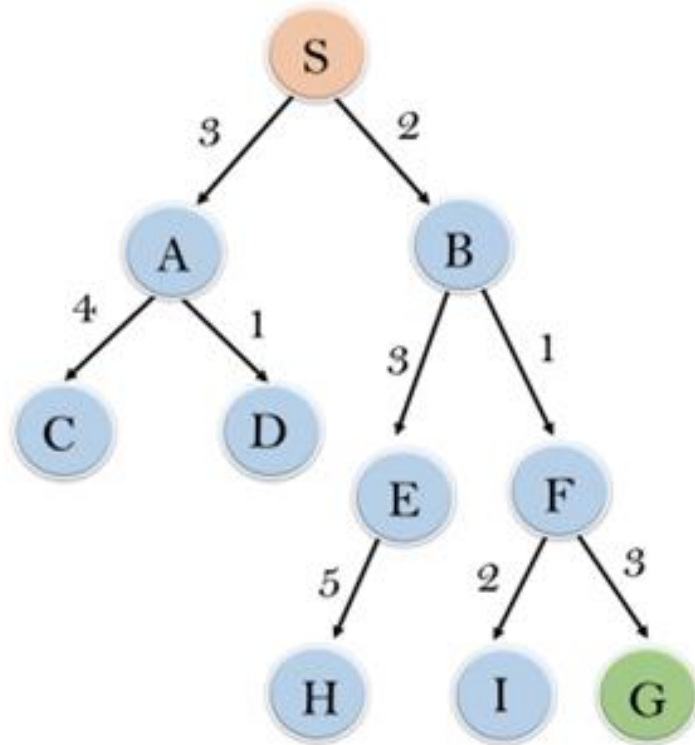
- 
- Step 1 : Put the initial node on a list START
- Step 2 : If (START is empty) or (START=GOAL) terminate search
- Step 3 : Remove the first node from START. Call this node  $a$
- Step 4 : If ( $a = \text{GOAL}$ ) terminate search with success
- Step 5 : Else if node  $a$  has successors, generate all of them. Find out how far they are from the goal node. Sort all the children generated so far by the remaining distance from the goal.
- Step 6 : Name this list as START 1
- Step 7 : Replace START with START 1
- Step 8 : Goto Step 2

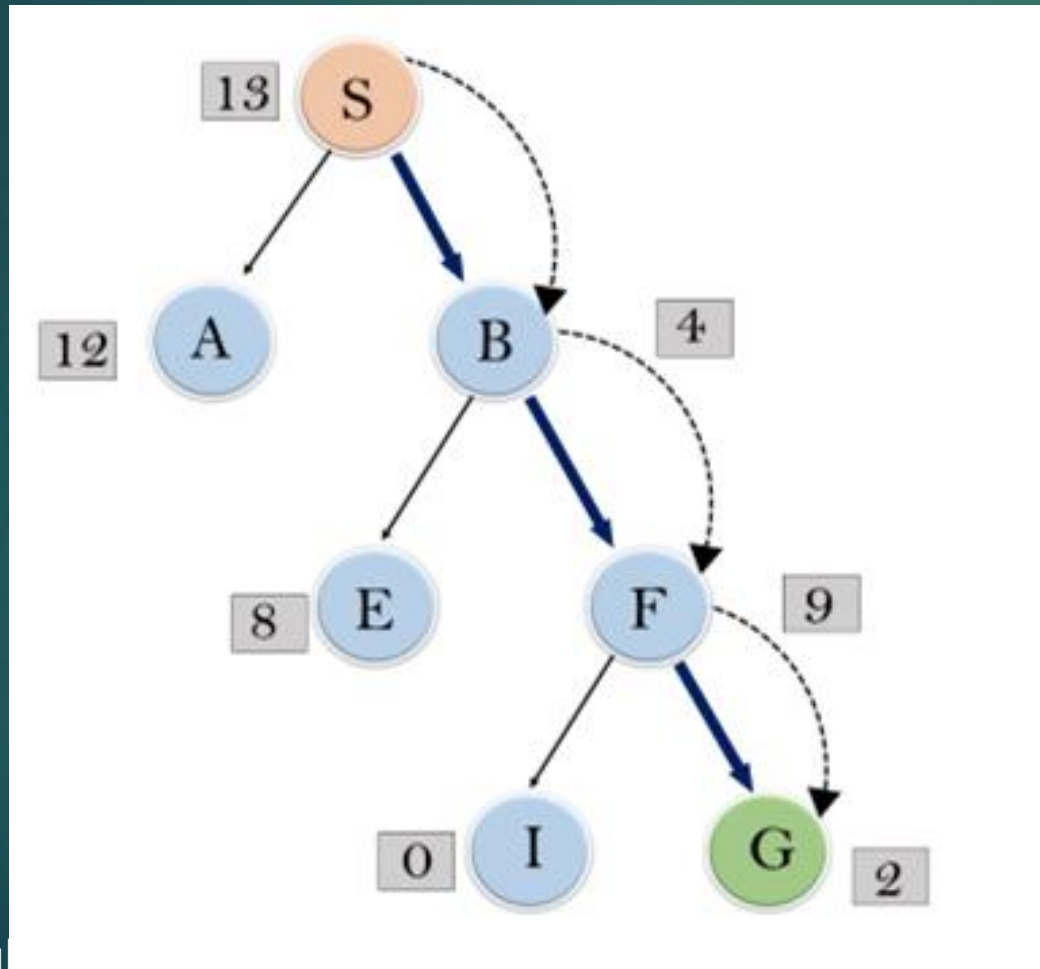
Fig. 3.9 Algorithm for best-first search

# Example



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

# Example



# Best-first Search Algorithm

- ▶ 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**

# Best-first Search Algorithm

## ► **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.

# Best-first Search Algorithm

- ▶ Time Complexity: The worst case time complexity of Greedy best first search is  $O(bm)$ .
- ▶ Space Complexity: The worst case space complexity of Greedy best first search is  $O(bm)$ . 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.

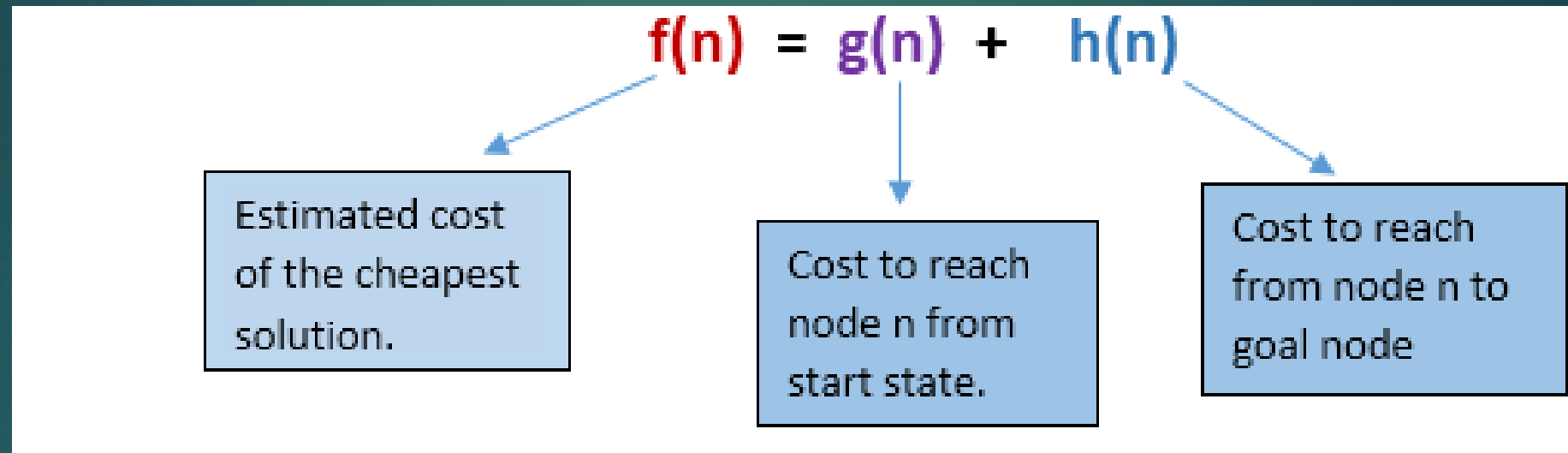


# A\* Search Algorithm

- ▶ 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  $g(n)$ .
- ▶ 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  $g(n)$ .



# A\* Search Algorithm



- ❑ 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

**Step 1.** Put the initial node on a list START.

**Step 2.** If (START is empty) or (START=GOAL), terminate search.

**Step 3.** Remove the first node from START, call it node a.

**Step 4.** If (a=GOAL) terminate search with success.

**Step 5.** Else if node a has successor, generate all of them. Estimate the fitness number of the successor by totalling the evaluation function value and cost function value. Sort the list by fitness number.

**Step 6.** Name the list as START 1 and replace it with list START.

**Step 7.** Go to step 2.

# Algorithm

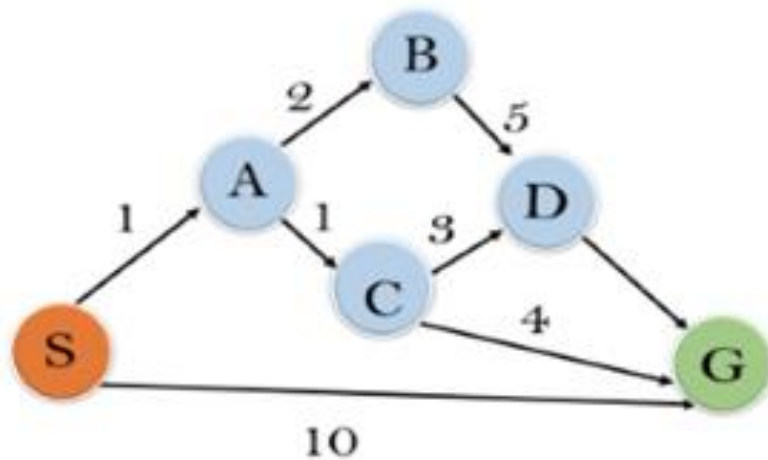
**Fig. 3.10 A Sample tree with fitness number used for A\* search**

- Step 1 : Put the initial node on a list START
- Step 2 : If (START is empty) or (START=GOAL) terminate search
- Step 3 : Remove the first node from START. Call this node  $a$
- Step 4 : If ( $a = \text{GOAL}$ ) terminate search with success
- Step 5 : Else if node  $a$  has successors, generate all of them. Estimate the fitness number of the successors by totaling the evaluation function value and the cost-function value. Sort the list by fitness number.
- Step 6 : Name the new list as START 1
- Step 7 : Replace START with START 1
- Step 8 : Goto Step 2

**Fig. 3.11 Algorithm for A\* algorithm method**

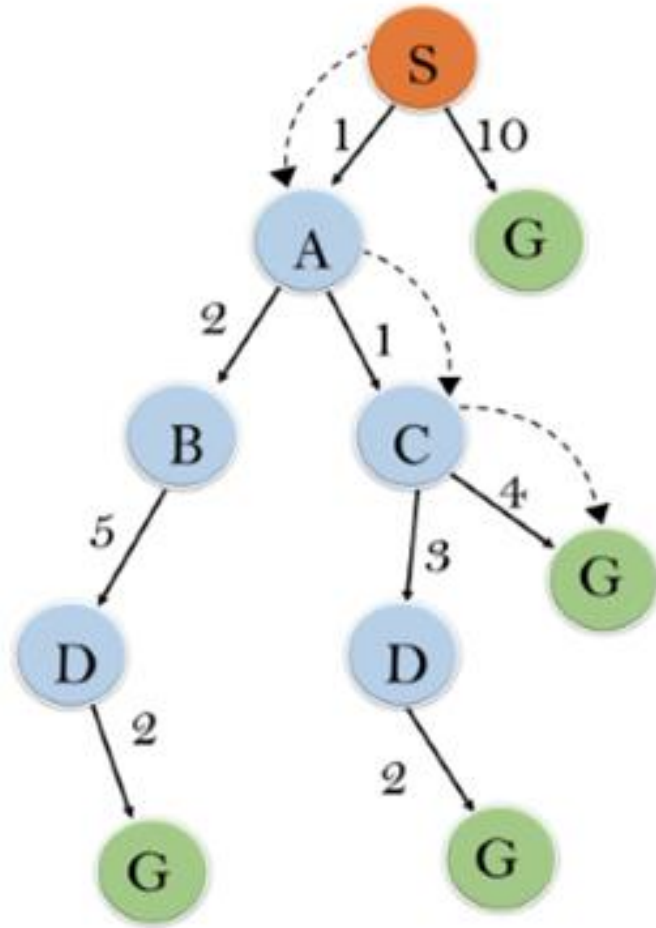
A very interesting observation about this algorithm

# Example



State	$h(n)$
S	5
A	3
B	4
C	2
D	6
G	0

# Solution



# Example

- ▶ Initialization:  $\{(S, 5)\}$
- ▶ Iteration1:  $\{(S \rightarrow A, 4), (S \rightarrow G, 10)\}$
- ▶ Iteration2:  $\{(S \rightarrow A \rightarrow C, 4), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$
- ▶ Iteration3:  $\{(S \rightarrow A \rightarrow C \rightarrow G, 6), (S \rightarrow A \rightarrow C \rightarrow D, 11), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$
- ▶ Iteration 4 will give the final result, as  $S \rightarrow A \rightarrow C \rightarrow G$  it provides the optimal path with cost 6.



# A\* Search Algorithm

## ► 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.



# 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  $f(n)$

- ▶ **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  $O(b^d)$

# Iterative deepening A\* (IDA\*)

- ▶ It perform depth first search with limited to some f-bound.
- ▶ Uses the formula:  $f(n) = h(n) + g(n)$
- ▶ Algorithm:
  - ▶ Perform depth-first search limited to some f-bound.
  - ▶ If goal found: OK
  - ▶ Else: increase the f-bound and restart.

# Small Memory $A^*$ ( $SMA^*$ )

- ▶ Like  $A^*$  search,  $SMA^*$  search is an optimal and complete algorithm for finding a least-cost path.
- ▶ Unlike  $A^*$ ,  $SMA^*$  will not run out of memory, unless the size of the shortest path exceeds the amount of space in available memory.
- ▶  $SMA^*$  addresses the possibility of running out of memory by pruning the portion of the search-space that is being examined.

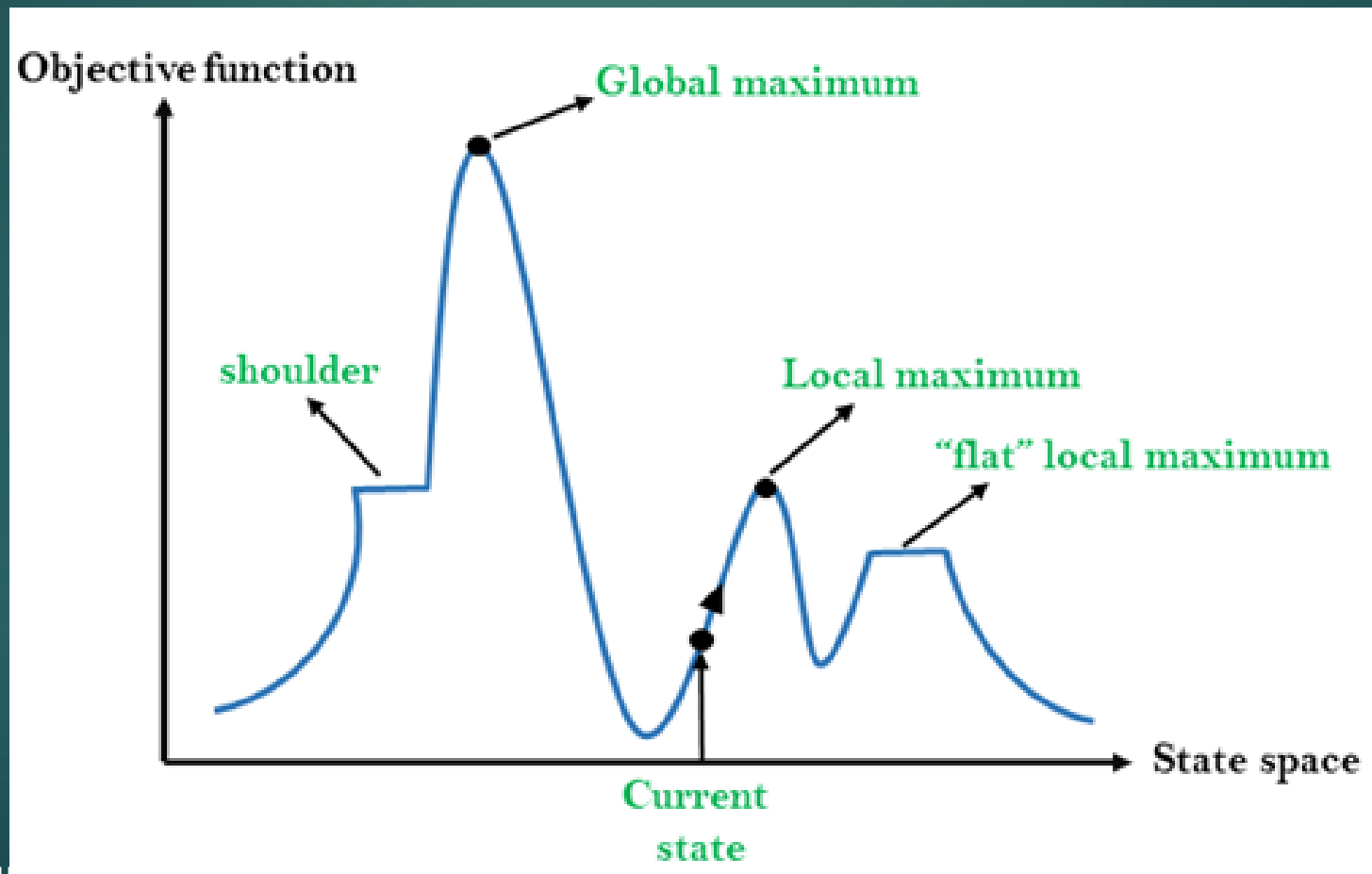
# Hill Climbing Algorithm

- ▶ 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.

# Features of Hill Climbing

- ▶ Following are some main features of Hill Climbing Algorithm:
  - ▶ Generate and Test variant: Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
  - ▶ Greedy approach: Hill-climbing algorithm search moves in the direction which optimizes the cost.
  - ▶ No backtracking: It does not backtrack the search space, as it does not remember the previous states.

# State-space Diagram for Hill Climbing





# State-space Diagram for Hill Climbing

- ▶ 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

- ▶ **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.
- ▶ **Step 3:** Select and apply an operator to the current state.
- ▶ **Step 4:** 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 5:** Exit.

# Algorithm

- Step 1 : Put the initial node on a list START
- Step 2 : If (START is empty) or (START=GOAL) terminate search
- Step 3 : Remove the first node from START. Call this node  $a$
- Step 4 : If ( $a=GOAL$ ) terminate search with success
- Step 5 : Else if node  $a$  has successors, generate all of them. Find out how far they are from the goal node, Sort them by the remaining distance from the goal and add them to the beginning of START.
- Step 6 : Goto Step 2.

**Fig. 3.5 Algorithm for hill-climbing procedure**

# Problems in Hill Climbing Algorithm

- ▶ **1. Local Maximum:**

- ▶ A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.
- ▶ **Solution:** Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



## ▶ 2. Plateau:

- ▶ A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.
- ▶ **Solution:** The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.



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

# Thank You

ANY QUERIES ??