UNIT – 2

18CSC305J / ARTIFICIAL INTELLIGENCE

Search

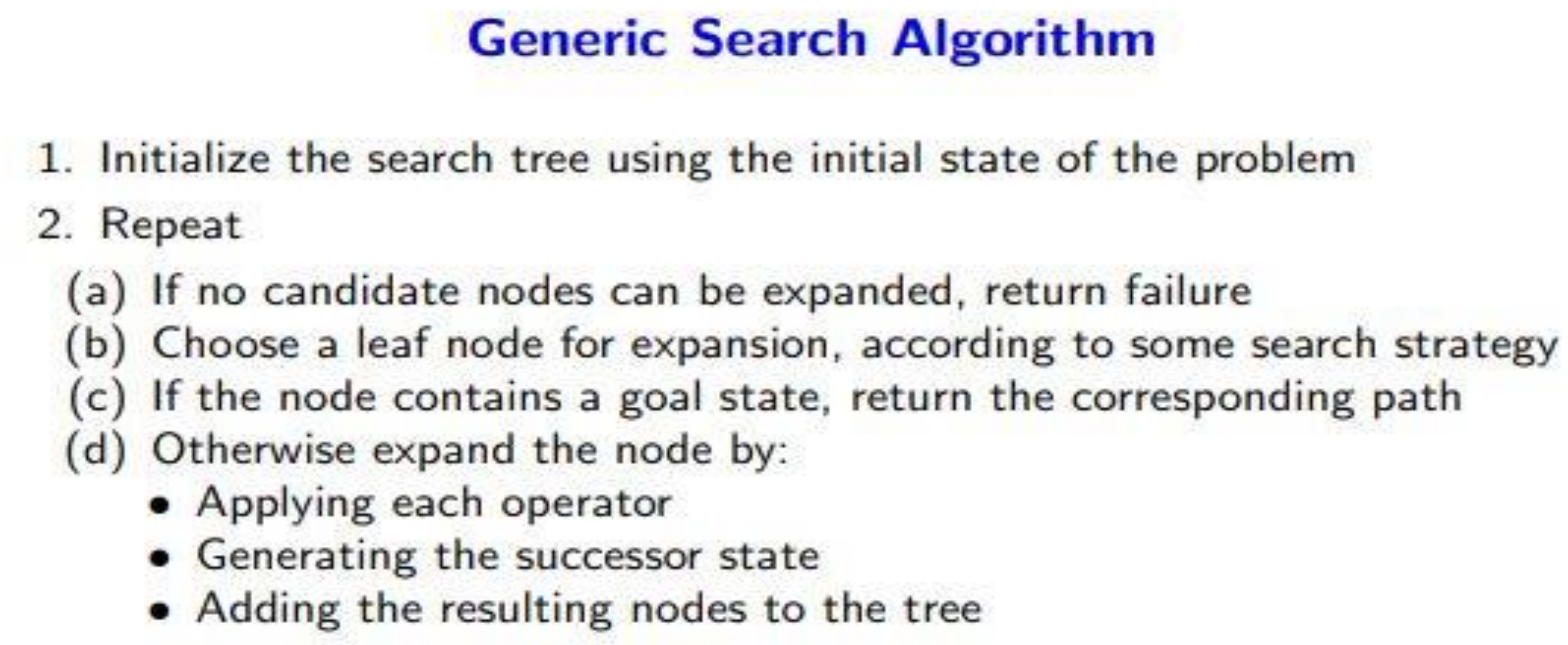
- Searching is the universal technique of problem solving in Artificial Intelligence
- A search problem consists of:
  - **A State Space.** Set of all possible states where you can be.
  - **A Start State.** The state from where the search begins.
  - **A Goal Test.** A function that looks at the current state returns whether or not it is the goal state.
  - The **Solution** to a search problem is a sequence of actions, called the **plan** that transforms the start state to the goal state.

Search Problem Components

- **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.

General Search Algorithm

- Basic idea:
  - simulated exploration of state space by generating successors of already-explored states (expanding states)



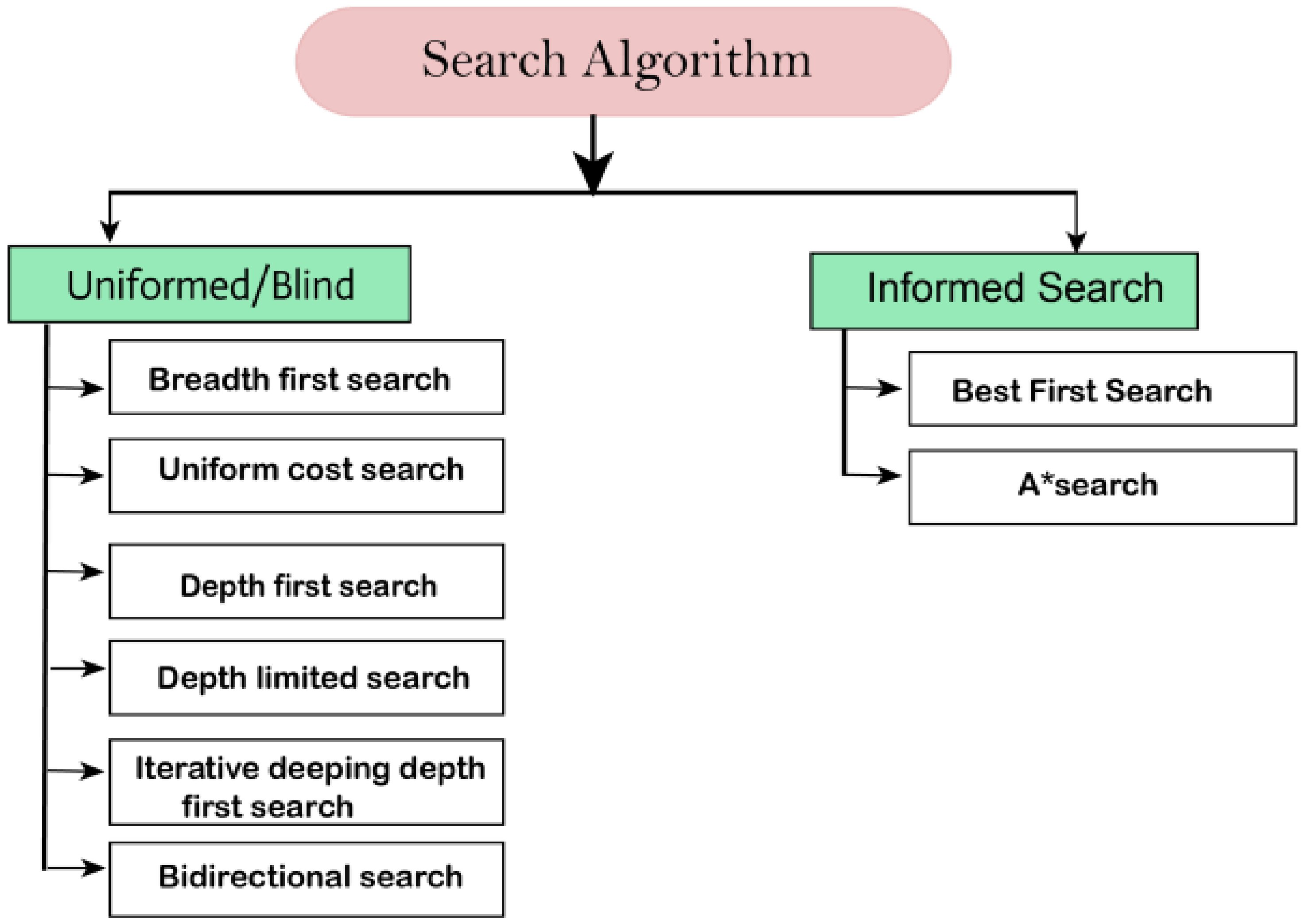
Generic Search Algorithm

1. Initialize the search tree using the initial state of the problem
2. Repeat
   (a) If no candidate nodes can be expanded, return failure
   (b) Choose a leaf node for expansion, according to some search strategy
   (c) If the node contains a goal state, return the corresponding path
   (d) Otherwise expand the node by:
   - Applying each operator
   - Generating the successor state
   - Adding the resulting nodes to the tree

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.
- Time and space complexity measured in terms of
    - b - maximum branching factor of the search tree
    - d - depth of a solution with minimal distance to root
    - m - maximum depth of the state space (may be infinity)

Types of search algorithms



Uninformed/Blind Search:

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

Uninformed/Blind Search

- **It can be divided into six main types:**
  - Breadth-first search
  - Uniform cost search
  - Depth-first search
  - Depth First Limited search
  - Iterative deepening depth-first search
  - Bidirectional Search

Informed Search

- Informed search algorithms use domain knowledge.
- In an informed search, problem information is available which can guide the search.
- Informed search strategies can find a solution more efficiently than an uninformed search strategy.
- **Informed search is also called a Heuristic search**.
- A heuristic is a way which might not always be guaranteed for **best solutions but guaranteed to find a good solution in reasonable time.**
- Informed search can solve much complex problem which could not be solved in another way.
- An example of informed search algorithms is a **traveling salesman problem.**

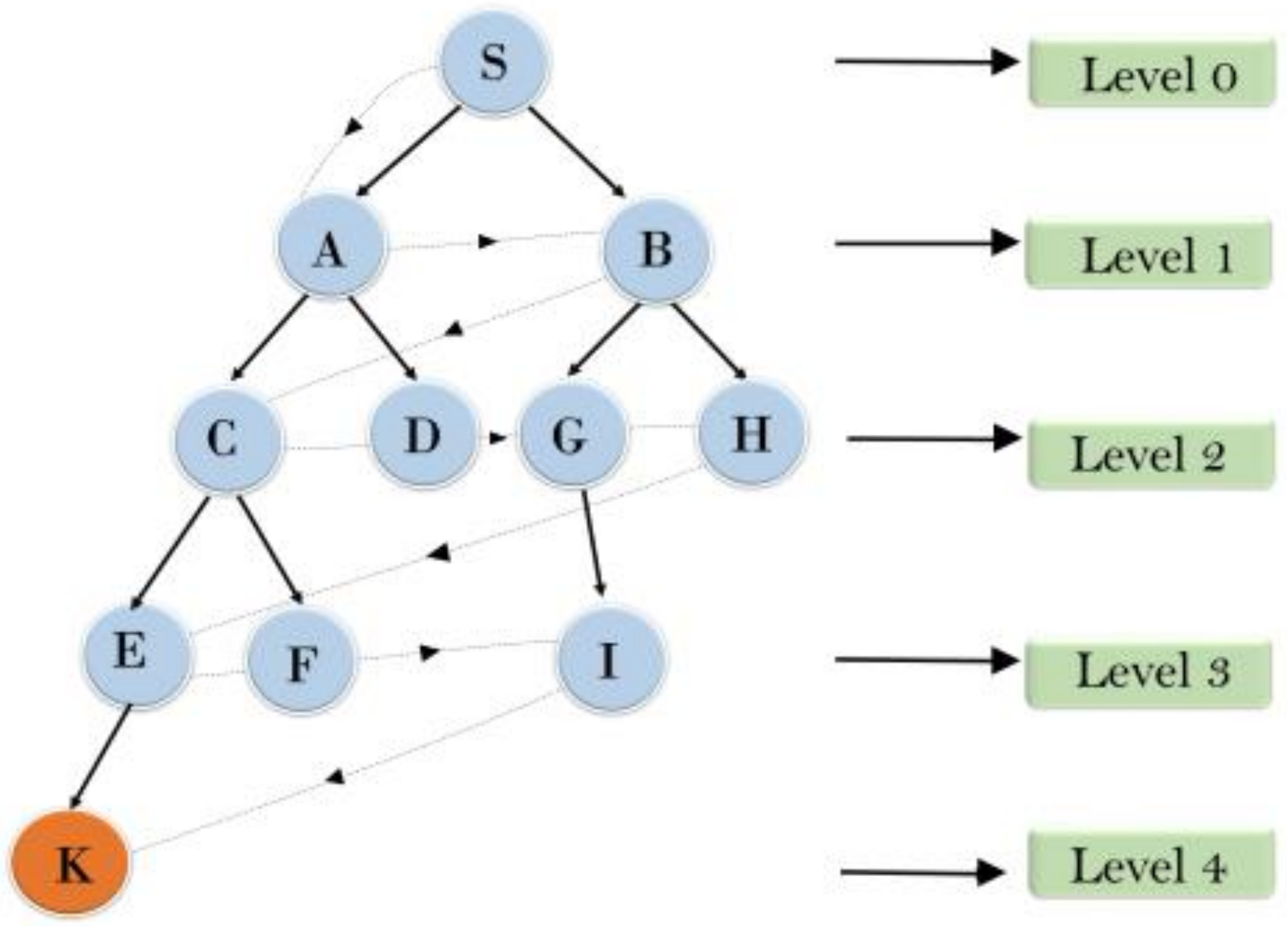
Types of Informed Search

- Greedy Search
- A* Search
- AO* algorithm
- Problem Reduction
- Hill climbing

1. Breadth-first Search

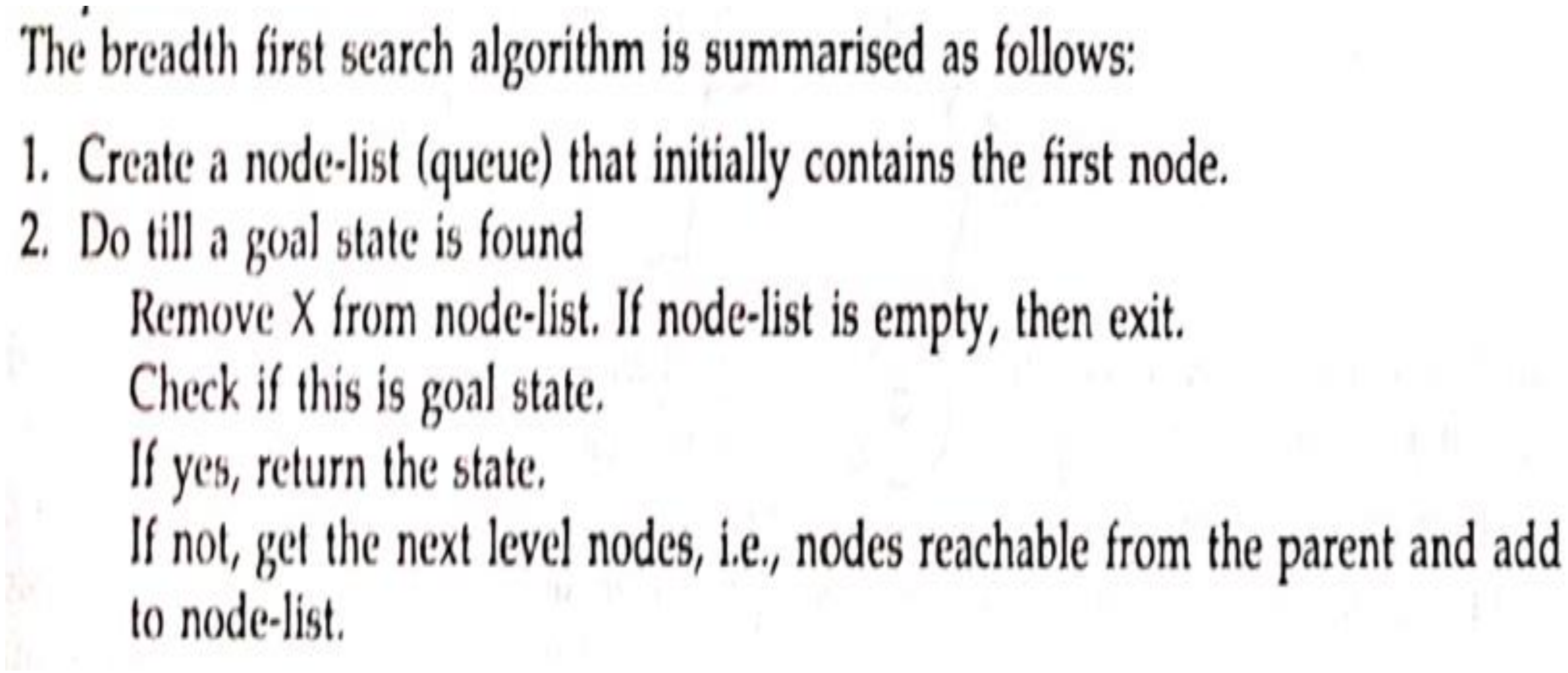
- Breadth-first search is the most common search strategy for traversing a tree or graph.
- This **algorithm searches breadthwise in a tree or graph,** so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level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

Breadth First Search



S- A –B – C –D – G-H- E-F- I-K

BFS Algorithm



The breadth first search algorithm is summarised as follows:

```
1. Create a node-list (queue) that initially contains the first node.
2. Do till a goal state is found
       Remove X from node-list. If node-list is empty, then exit.
       Check if this is goal state.
       If yes, return the state.
       If not, get the next level nodes, i.e., nodes reachable from the parent and add
       to node-list.
```

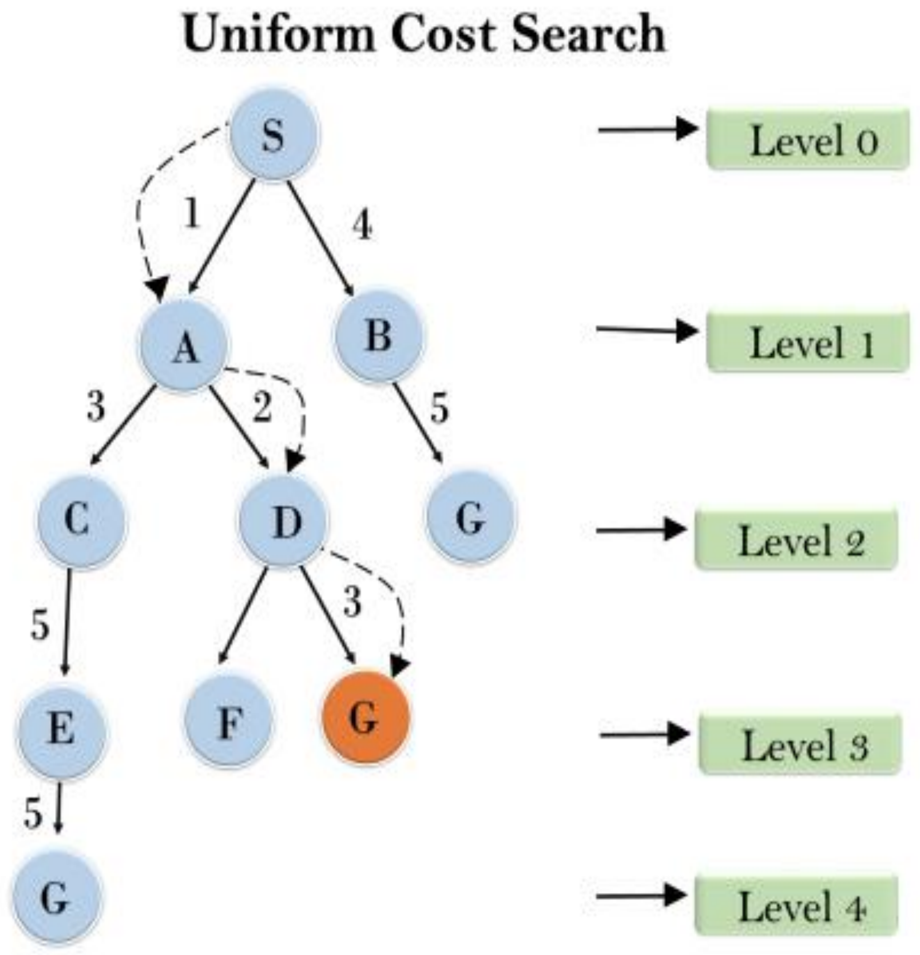
- **Time Complexity:** Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.
  - **$T(b) = 1+b+b^2+b^3+.......+ b^d= O(b^d)$**
- **Space Complexity:** Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.
- **Completeness:** BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.
- **Optimality:** BFS is optimal if path cost is a non-decreasing function of the depth of the node

- **Disadvantage**
  - Space is the big problem

2. Uniform Cost search

- Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph.
- This algorithm comes into play when a different cost is available for each edge.
- The primary goal of the uniform-cost search is to find a path to the goal node which has the **lowest cumulative cost**.
- Uniform-cost search expands nodes according to their path costs form the root node.
- It can be used to solve any graph/tree where the optimal cost is in demand.
- A uniform-cost search algorithm is implemented by the **priority queue.** It gives maximum priority to the lowest cumulative cost.
- Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

EXAMPLE


Uniform Cost Search
S
1
4
A
B
3
2
5
C
D
G
5
3
E
F
G
5
G
Level 0
Level 1
Level 2
Level 3
Level 4

Advantages:

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

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

- **Completeness:**
  - Uniform-cost search is complete, such as if there is a solution, UCS will find it.
- **Time Complexity:**
  - Let C* **is Cost of the optimal solution**, and **ε** is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε.
  - Hence, the worst-case time complexity of Uniform-cost search is**$O(b^{1 + [C^*/\varepsilon]})$**/.
- **Space Complexity:**
  - The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is **$O(b^{1 + [C^*/\varepsilon]})$**.
- **Optimal:**
  - Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

3. Depth-first Search

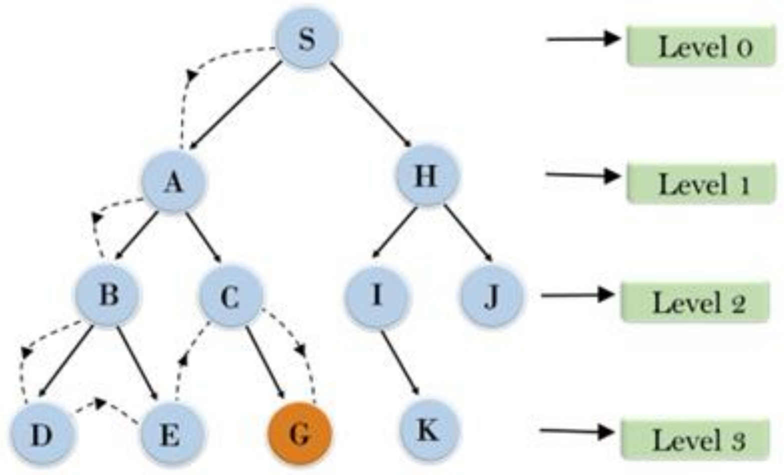
- Depth-first search is **a recursive algorithm** for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- Depth first search (DFS) algorithm starts with the initial node of the graph G, and then goes to deeper and deeper until the **goal node** or the node which has no children.
- The algorithm, then **backtracks** from the dead end towards the most recent node that is yet to be completely unexplored.
- DFS uses a **stack data structure** for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Example:

- In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:
- **Root node--->Left node ----> right node.**
- It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.

- S- A- B- E-C-G-H-I-K-J

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 will be equivalent to the node traversed by the algorithm. It is given by:
- **$T(n)= 1+ n^2+ n^3 +.........+ n^m=O(n^m)$**
- **Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)**
- **Space Complexity:** DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.
- **Optimal:** DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node

Depth-Limited Search (DLS):

- A depth-limited search is similar to depth-first search with a **predetermined limit.**
- Depth-limited search can solve the drawback of the infinite path in the Depth-first search.
- In this algorithm, the node at the depth limit will treat as it has no successor nodes further.
- Depth-limited search can be terminated with two Conditions of failure:
  - **Standard failure value:** It indicates that problem does not have any solution.
  - **Cut off failure value:** It defines no solution for the problem within a given depth limit.

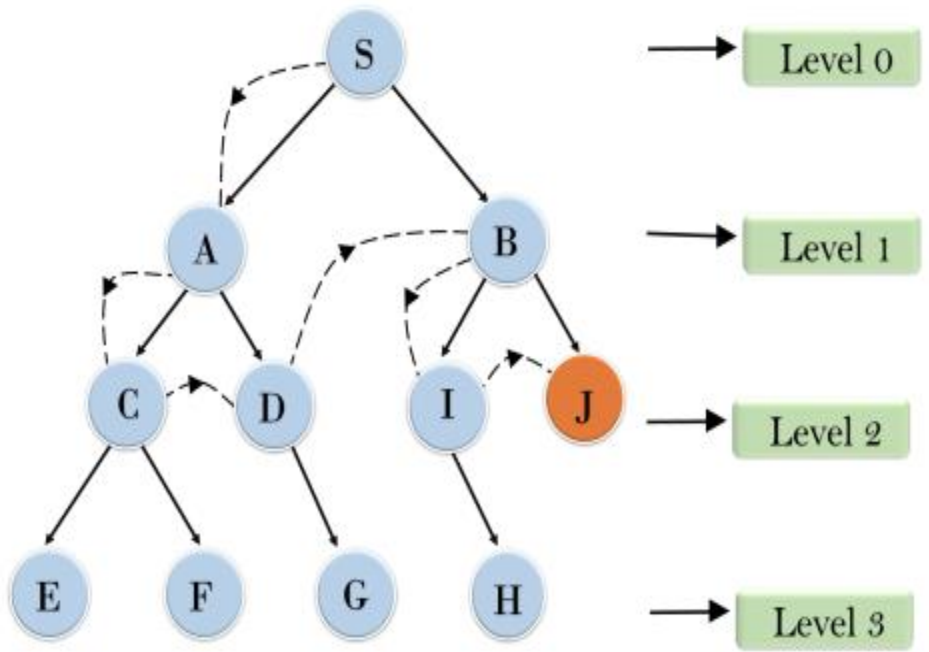
Advantages:

- Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Depth Limited Search



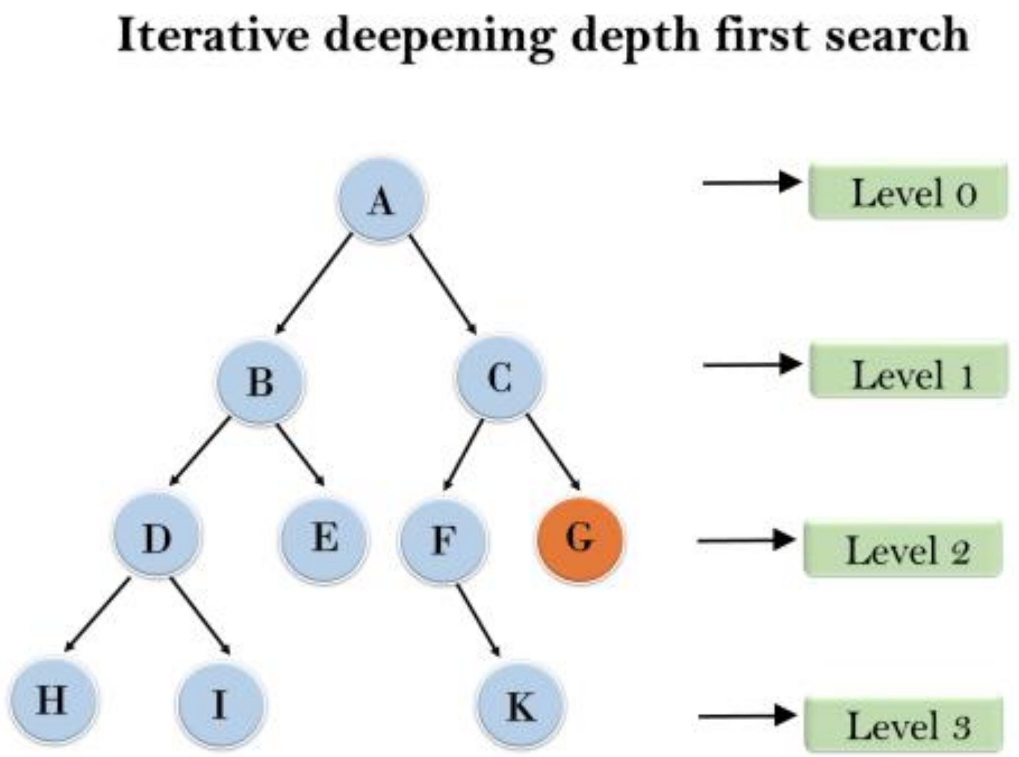
- **Completeness:** DLS search algorithm is complete if the solution is above the depth-limit.
- **Time Complexity:** Time complexity of DLS algorithm is $\mathbf{O(b^{\ell})}$.
- **Space Complexity:** Space complexity of DLS algorithm is O$\mathbf{(b\times\ell)}$.
- **Optimal:** Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $\ell>d$.

Iterative deepening depth-first Search:

- The iterative deepening algorithm is a combination of DFS and BFS algorithms.
- This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.
- This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.
- This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.
- The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.
- **Advantages:**
- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.
- **Disadvantages:**
- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:

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



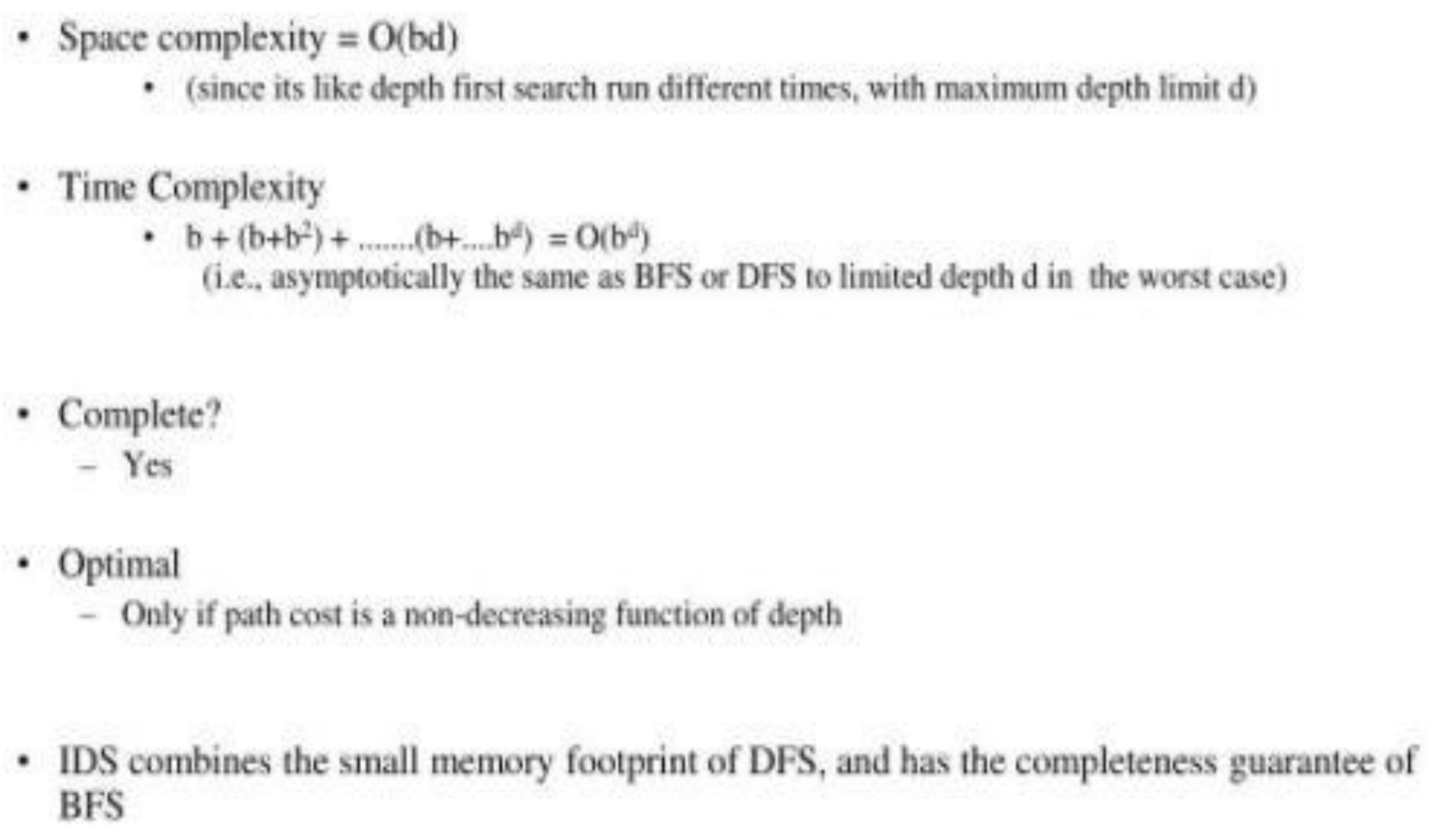
1'st Iteration-----> A

2'nd Iteration----> A, B, C

3'rd Iteration------>A, B, D, E, C, F, G

4'th Iteration------>A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.



- Space complexity = O(bd)
  - (since its like depth first search run different times, with maximum depth limit d)

- Time Complexity
  - $b + (b+b^2) + ...(b+....b^d) = O(b^d)$
    (i.e., asymptotically the same as BFS or DFS to limited depth d in the worst case)

- Complete?
  – Yes

- Optimal
  – Only if path cost is a non-decreasing function of depth

- IDS combines the small memory footprint of DFS, and has the completeness guarantee of BFS

Bidirectional Search Algorithm:

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

Advantages:

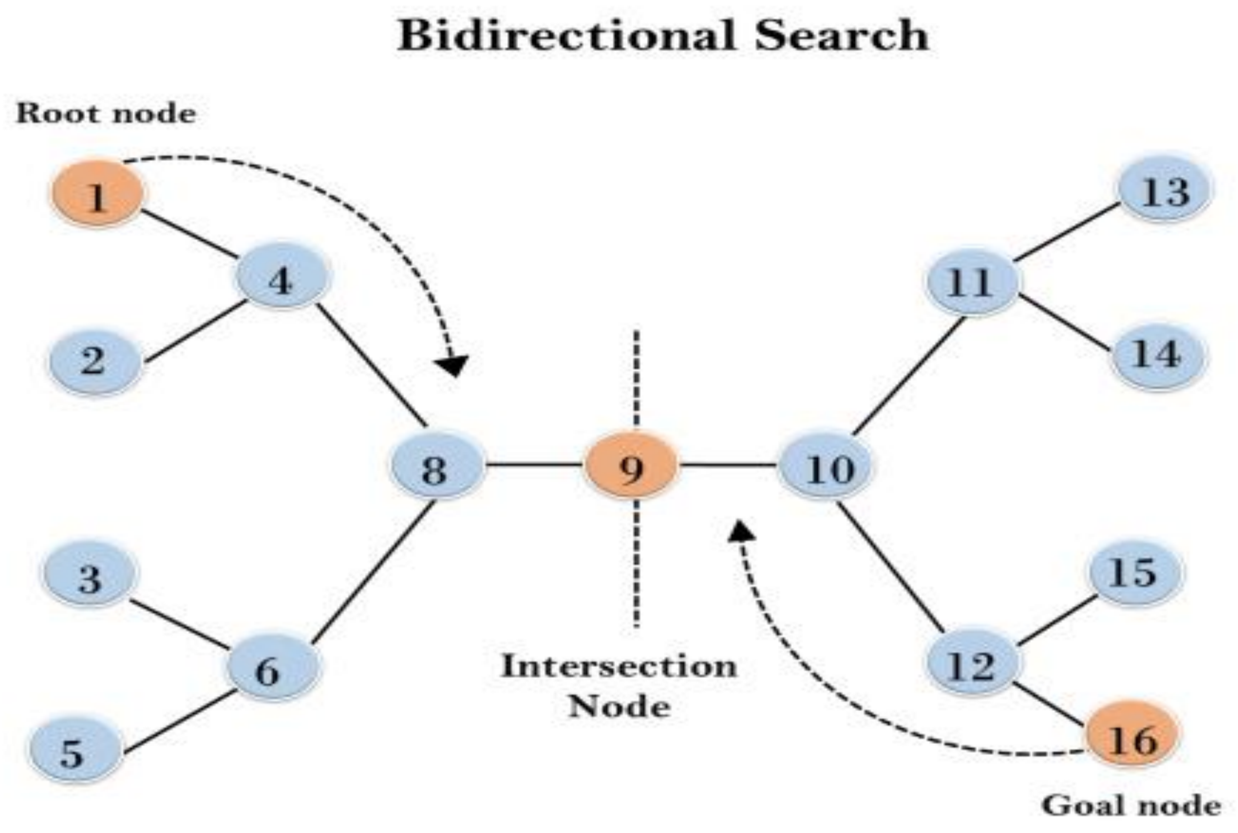
- Bidirectional search is fast.
- Bidirectional search requires less memory

Disadvantages:

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

Example:

- In the below search tree, bidirectional search algorithm is applied.
- This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.
- The algorithm terminates at node 9 where two searches meet.



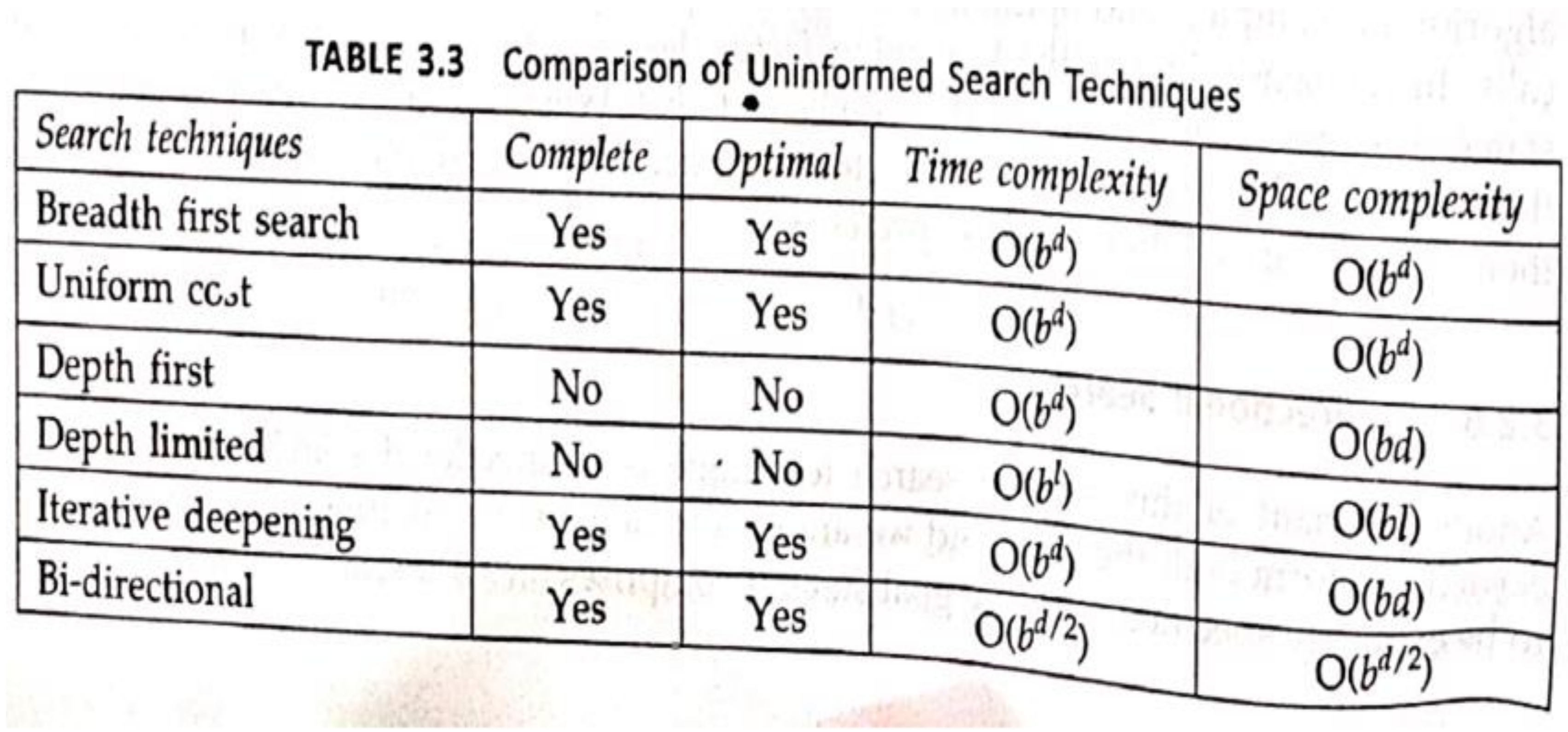


TABLE 3.3 Comparison of Uninformed Search Techniques

Search techniques	*Complete*	*Optimal*	*Time complexity*	*Space complexity*
Breadth first search	Yes	Yes	$O(b^d)$	$O(b^d)$
Uniform cost	Yes	Yes	$O(b^d)$	$O(b^d)$
Depth first	No	No	$O(b^d)$	$O(bd)$
Depth limited	No	No	$O(b^l)$	$O(bl)$
Iterative deepening	Yes	Yes	$O(b^d)$	$O(bd)$
Bi-directional	Yes	Yes	$O(b^{d/2})$	$O(b^{d/2})$

```
        S ... Initial State
       /|\
     1/ 5 \8
     /  |  \
    A   B   C
   /|\  |  /
 3/ 7 9 4 /5
 /  |  \|/
D   E   G .... Goal State
```

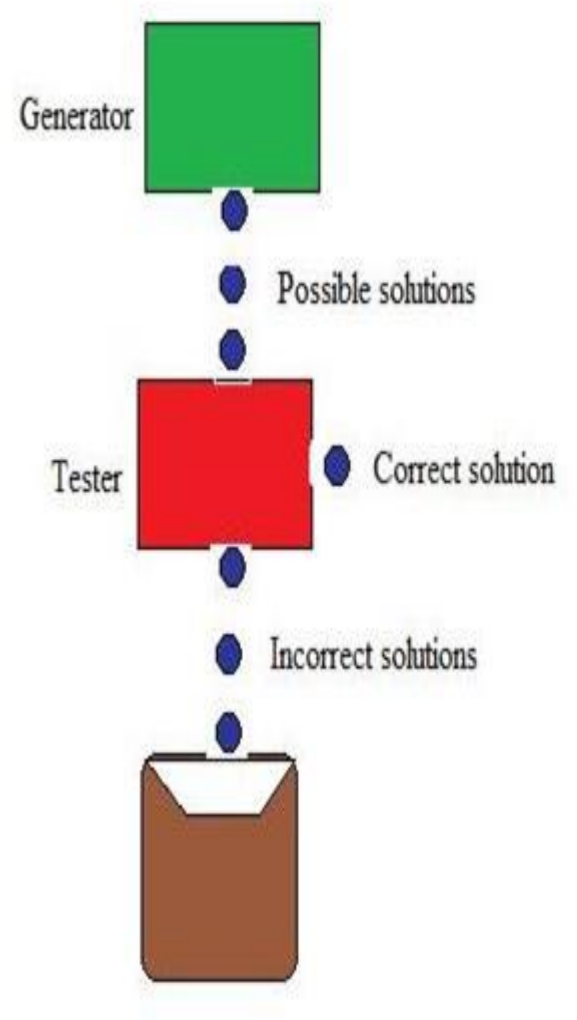
Informed Search

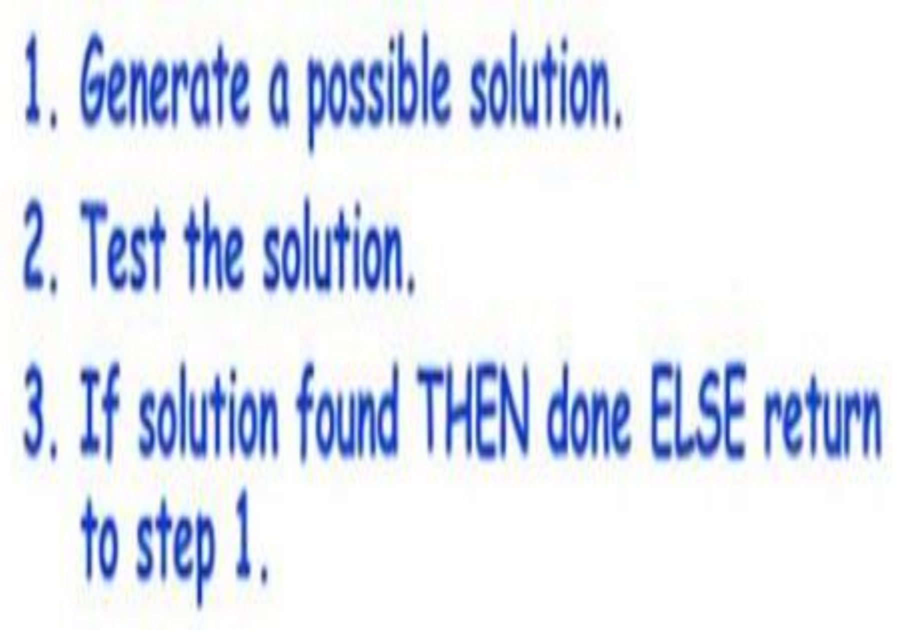
- Informed search technique uses the idea of heuristic, so it is also called Heuristic search technique.
- This technique uses the information about the domain, or knowledge about the problem and scenario to move to the goal state.
- These methods are usefull in solving the tough problems.The classic example is travelling salesman problem.
- **Heuristic Function:**
  - It is the one that guides decision of selection of a path.
  - h(n) provides an estimate of the cost of the path from the given node to reach to the closest goal node.

Generate and Test

- Generate and test search algorithm is a very simple algorithm that guarantees to find a solution if done systematically and there exists a solution.
- The generate and Test algorithm is a depth first search procedure because complete possible solutions are generated before test.
- This can be implemented states are likely to appear often in a tree
- It can be implemented on a search graph rather than a tree

Generate and Test





1. Generate a possible solution.
2. Test the solution.
3. If solution found THEN done ELSE return to step 1.

Best First Search

- Best first search is a traversal technique that decides which node is to be visited next by checking which node is the **most promising one** and then check it.
- Best first search is an instance of graph search algorithm in which a node is selected for expansion based on evaluation function f(n). here, f(n)=h(n).
- Best first search can be implemented a priority queue, a data structure that will maintain the fringe in ascending order of f values.

OR Graph

- The notion of OR graphs is required in order to avoid the revisiting of paths and for propagating back to the successor, in case it is required.
- For the algorithm to proceed, a node contains the following information:
  1. Description of the ste it represents.
  2. Indication how promising it is.
  3. Parent link that points to the best node it has reached from.
  4. List of nodes that are generated from it.
- A Graph employing these descriptions is called OR graph.

- To have this kind of graph search, two lists of node is required,

1. Open List:

- It consist of list of nodes that have been generated and on whome the heuristic function has already been applied, but yet not examined.

2. Closed List:

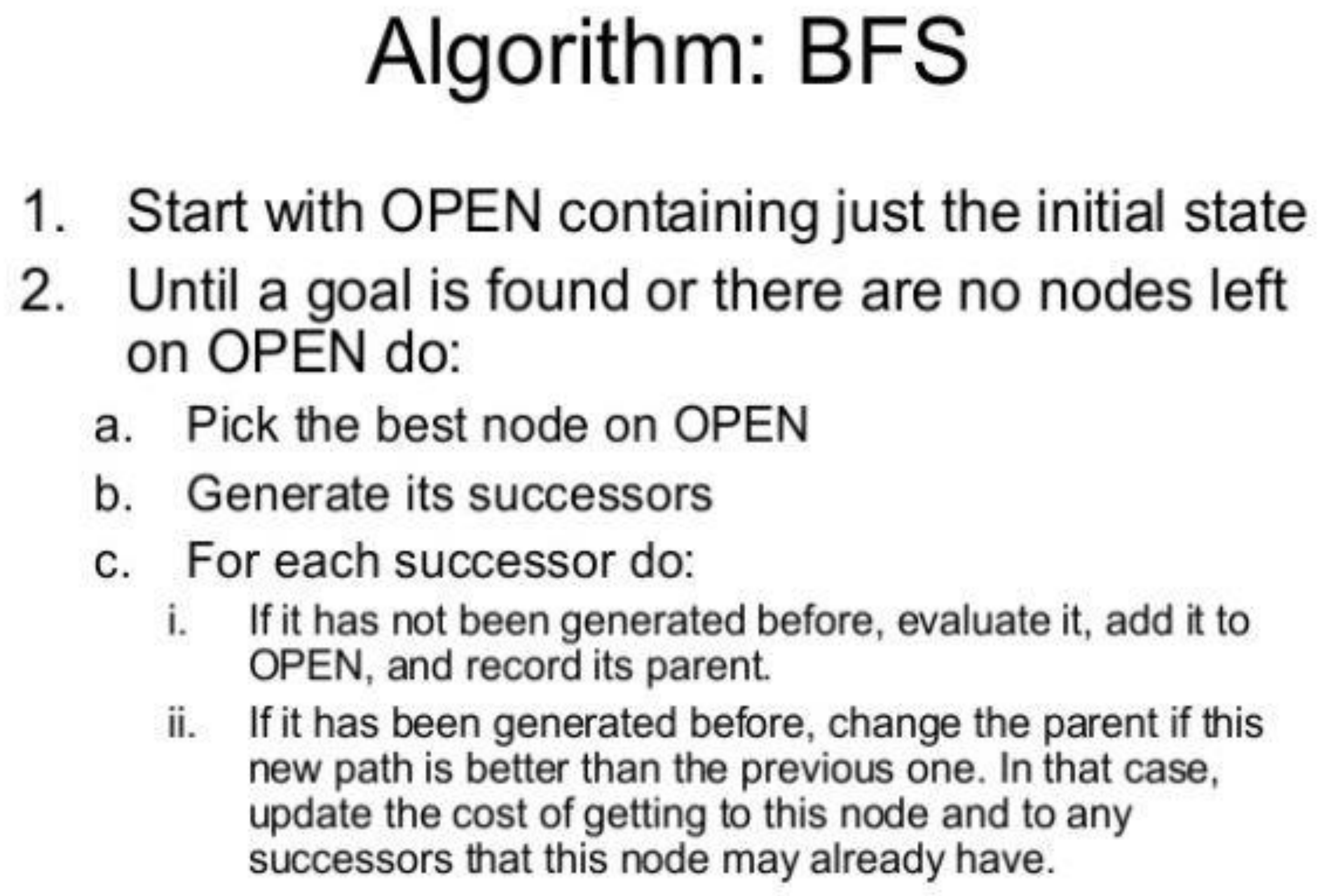
- It contains the nodes that have already been examined.

f(n) is the function that can be defined as the sum of two elements,

  - the cost of reaching from the initial node to the current node g(n) and
  - the additional cost of getting from the current node to the goal state, h(n).

  so, f()=g(n)+h(n) defines the function.

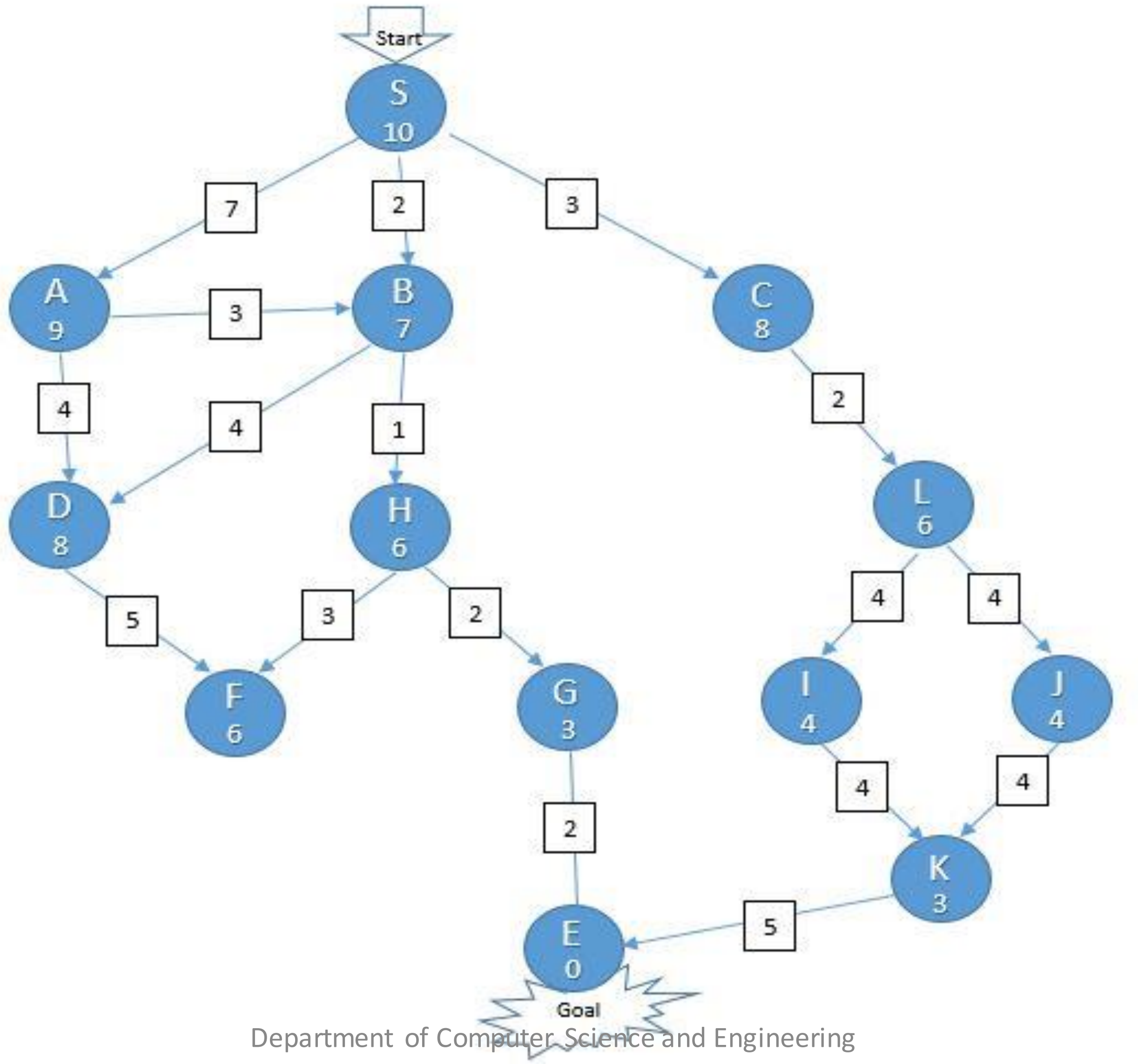
Best First Search - Algorithm



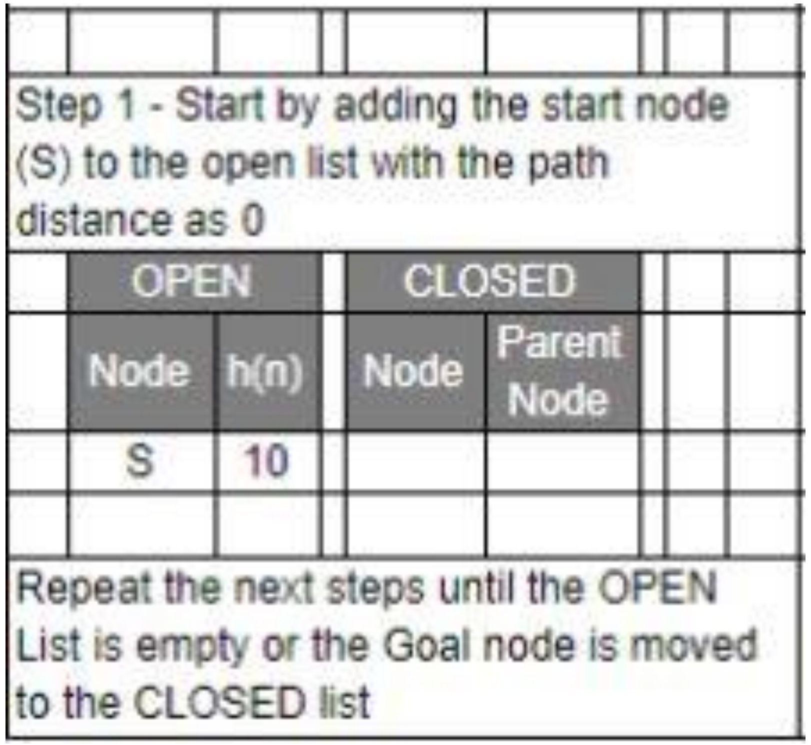
Algorithm: BFS

1. Start with OPEN containing just the initial state
2. Until a goal is found or there are no nodes left on OPEN do:
   a. Pick the best node on OPEN
   b. Generate its successors
   c. For each successor do:
      i. If it has not been generated before, evaluate it, add it to OPEN, and record its parent.
      ii. If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

BFS - Example


Start
S 10
7
2
3
A 9
3
B 7
C 8
4
4
1
2
D 8
H 6
L 6
5
3
2
4
4
F 6
G 3
I 4
J 4
2
4
4
K 3
5
E 0
Goal

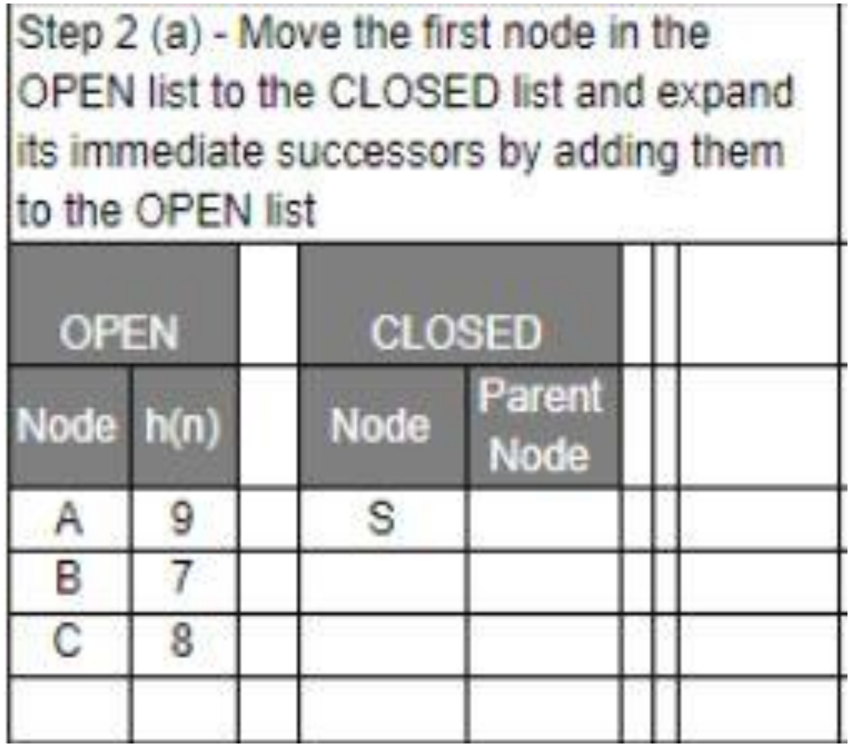
BFS - Working



Step 1 - Start by adding the start node (S) to the open list with the path distance as 0

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
S	10		

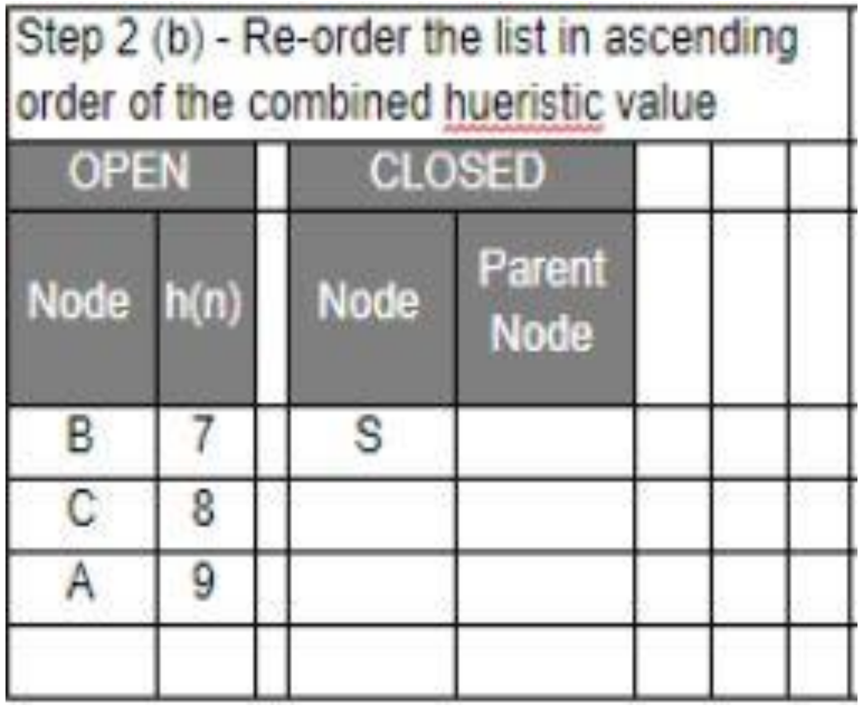
Repeat the next steps until the OPEN List is empty or the Goal node is moved to the CLOSED list



Step 2 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

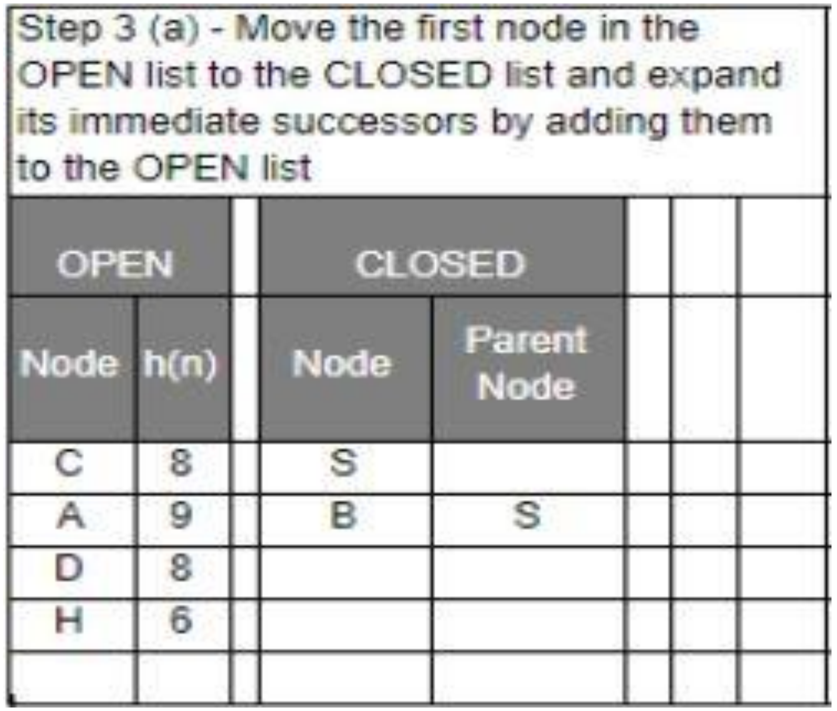
OPEN		CLOSED	
Node	h(n)	Node	Parent Node
A	9	S	
B	7		
C	8		

BFS - Working



Step 2 (b) - Re-order the list in ascending order of the combined hueristic value

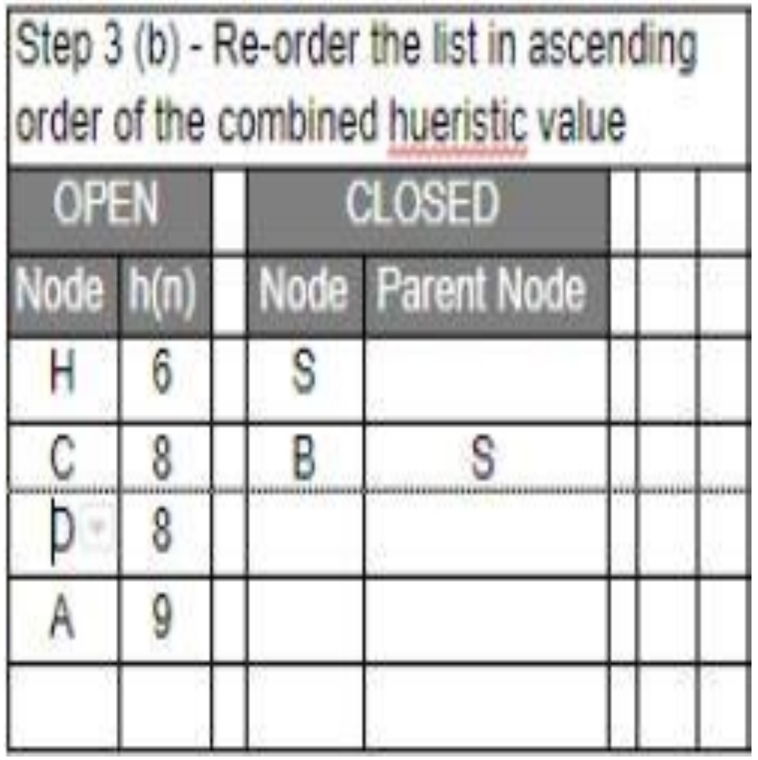
OPEN			CLOSED				
Node	h(n)		Node	Parent Node			
B	7		S				
C	8						
A	9						



Step 3 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

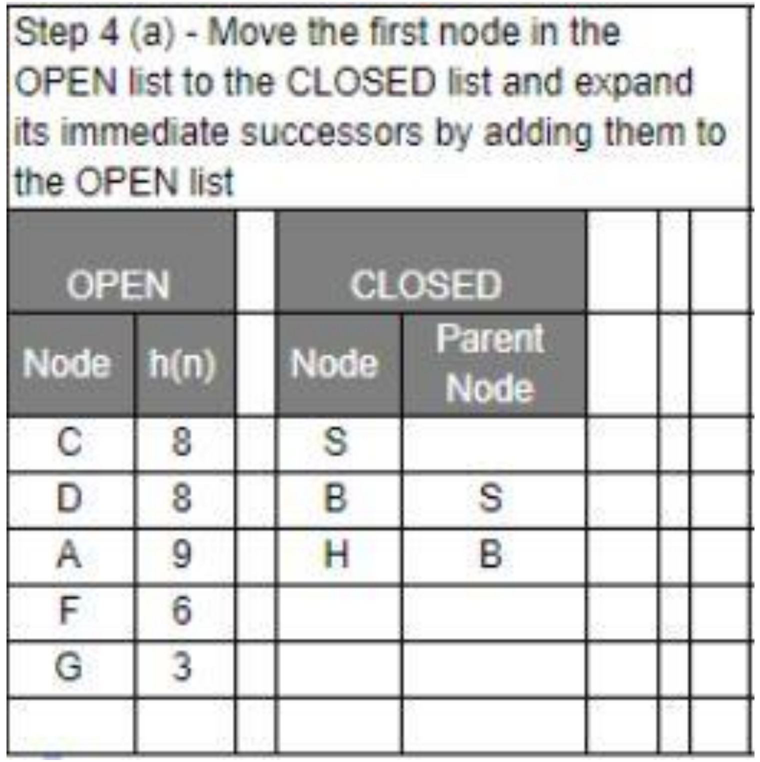
OPEN			CLOSED				
Node	h(n)		Node	Parent Node			
C	8		S				
A	9		B	S			
D	8						
H	6						

BFS - Working



Step 3 (b) - Re-order the list in ascending order of the combined hueristic value

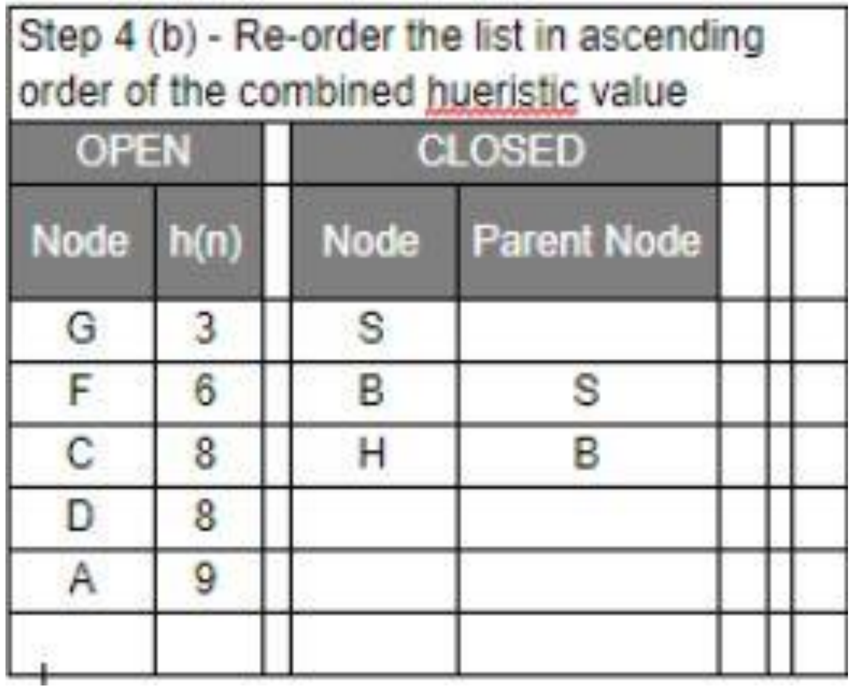
OPEN		CLOSED	
Node	h(n)	Node	Parent Node
H	6	S	
C	8	B	S
D	8		
A	9		



Step 4 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

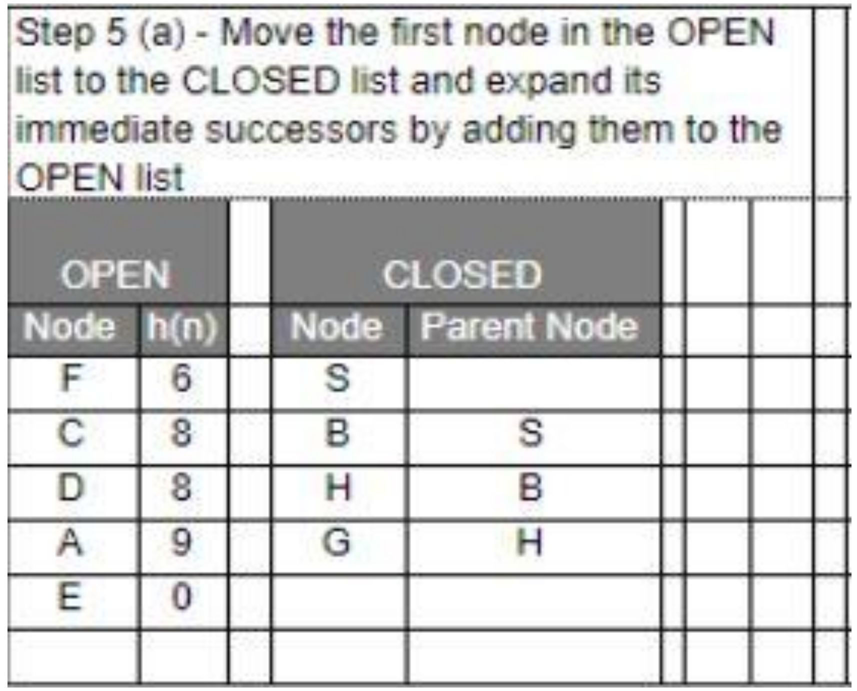
OPEN		CLOSED	
Node	h(n)	Node	Parent Node
C	8	S	
D	8	B	S
A	9	H	B
F	6		
G	3		

BFS - Working



Step 4 (b) - Re-order the list in ascending order of the combined hueristic value

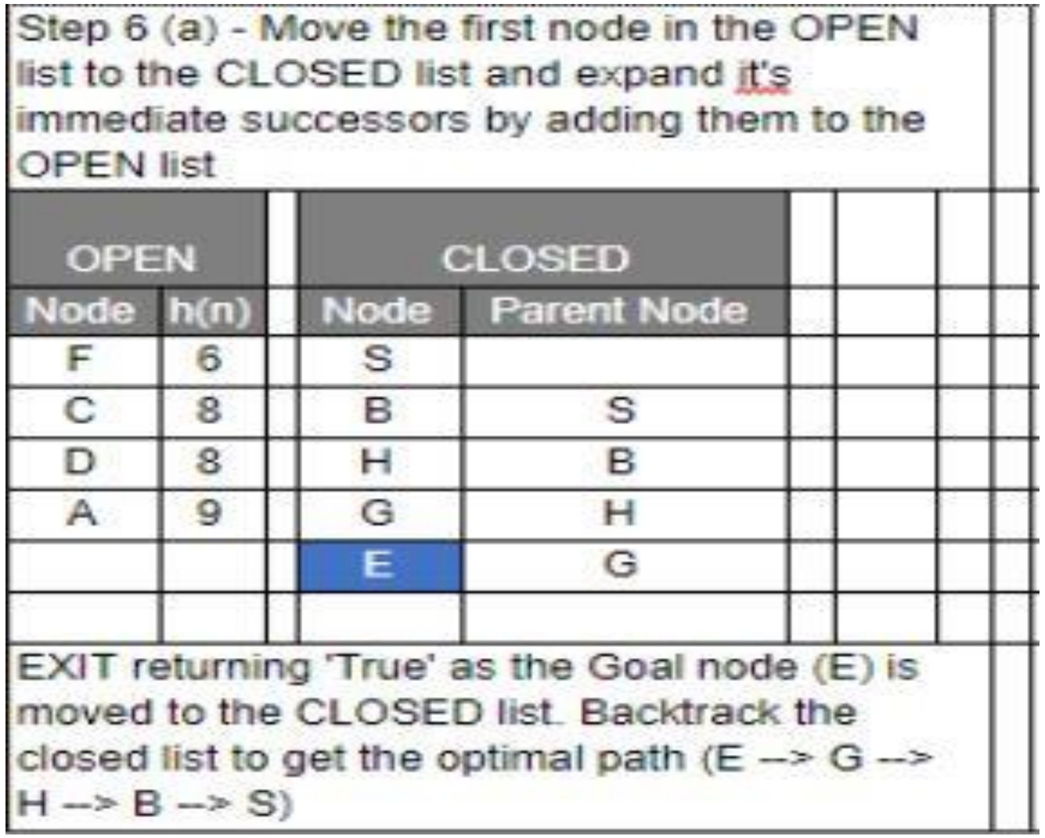
OPEN		CLOSED	
Node	h(n)	Node	Parent Node
G	3	S	
F	6	B	S
C	8	H	B
D	8		
A	9		



Step 5 (a) - Move the first node in the OPEN list to the CLOSED list and expand its immediate successors by adding them to the OPEN list

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
F	6	S	
C	8	B	S
D	8	H	B
A	9	G	H
E	0		

BFS - Working



Step 6 (a) - Move the first node in the OPEN list to the CLOSED list and expand it's immediate successors by adding them to the OPEN list

OPEN		CLOSED	
Node	h(n)	Node	Parent Node
F	6	S	
C	8	B	S
D	8	H	B
A	9	G	H
		E	G

EXIT returning 'True' as the Goal node (E) is moved to the CLOSED list. Backtrack the closed list to get the optimal path (E --> G --> H --> B --> S)

<u>S --> B --> H --> G --> E</u>

Time and space complexities are O(b^m) where m represents thaximum depth.

*A** Search

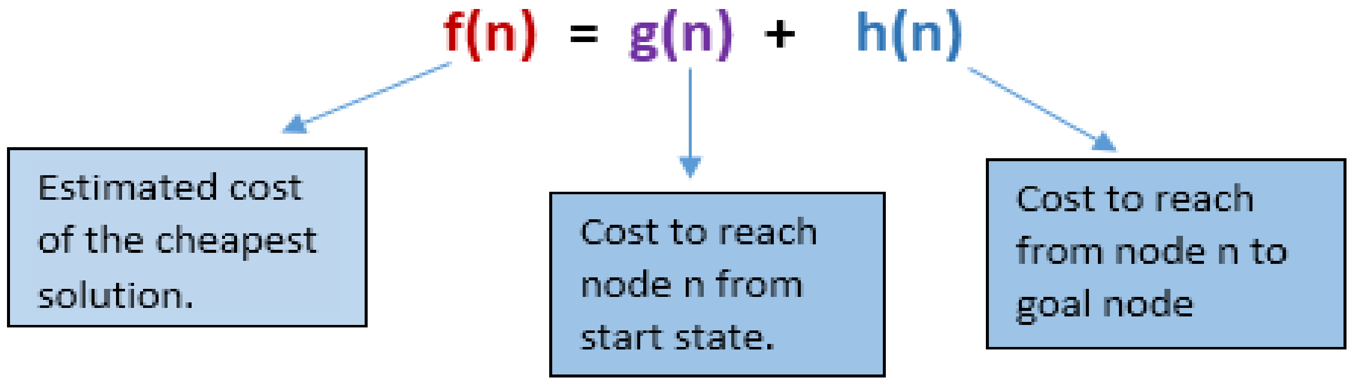
- It is an informed search algorithm, as it uses information about **path cost** and also uses **heuristics** to find the solution.
- To find the shortest path between nodes or graphs
- **A * algorithm** is a searching algorithm that searches for the shortest path between the *initial and the final state*
- It is an **advanced BFS** that *searches for shorter paths first rather than the longer paths*.
- A* is **optimal** as well as a **complete** algorithm
- This algorithm is presented by Hart.
- Evaluation Function f(n) = g(n)+h(n).

A* Search

- It calculates the cost, f(n) (n being the neighboring node), to travel to all of the neighboring nodes, and then enters the node with the lowest value of f(n).
- These values are calculated with the following formula:

$$f(n)=g(n)+h(n)$$

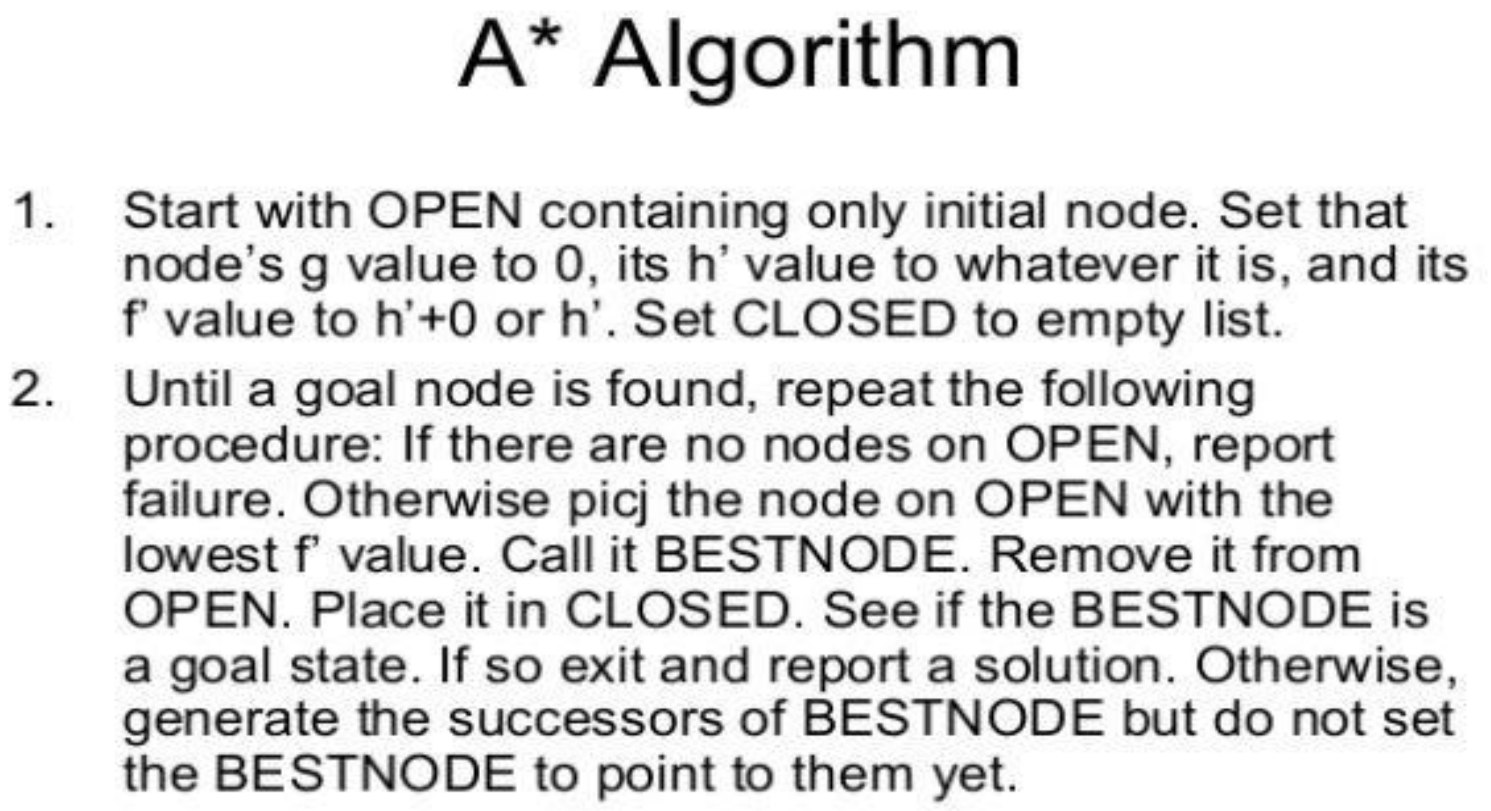
- g(n) being the value of the shortest path from the start node to node *n*, and h(n) being a heuristic approximation of the node's value.


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

Heuristic value h(n)

- A given heuristic function h(n) is ***admissible*** if it never overestimates the real distance between *n* and the goal node.
  - Therefore, for every node *n*, **h(n)≤ h*(n)**
    - h*(n) being the real distance between *n* and the goal node.
- A given heuristic function h(n) is ***consistent*** if the estimate is always less than or equal to the estimated distance between the goal *n* and any given neighbor, plus the estimated cost of reaching that neighbor: cost(n,m)+h(m)≥h(n)
  - c(n,m) being the distance between nodes n and m.
- Additionally, if h(n) is consistent, then we know the optimal path to any node that has been already inspected. This means that this function is ***optimal***.

A* Search Algorithm



A* Algorithm

1. Start with OPEN containing only initial node. Set that node's g value to 0, its h' value to whatever it is, and its f' value to h'+0 or h'. Set CLOSED to empty list.
2. Until a goal node is found, repeat the following procedure: If there are no nodes on OPEN, report failure. Otherwise picj the node on OPEN with the lowest f' value. Call it BESTNODE. Remove it from OPEN. Place it in CLOSED. See if the BESTNODE is a goal state. If so exit and report a solution. Otherwise, generate the successors of BESTNODE but do not set the BESTNODE to point to them yet.

A* Algorithm (contd)

- For each of the SUCCESSOR, do the following:

a. Set SUCCESSOR to point back to BESTNODE. These backwards links will make it possible to recover the path once a solution is found.

b. Compute g(SUCCESSOR) = g(BESTNODE) + the cost of getting from BESTNODE to SUCCESSOR

c. See if SUCCESSOR is the same as any node on OPEN. If so call the node OLD.

d. If SUCCESSOR was not on OPEN, see if it is on CLOSED. If so, call the node on CLOSED OLD and add OLD to the list of BESTNODE's successors.

e. If SUCCESSOR was not already on either OPEN or CLOSED, then put it on OPEN and add it to the list of BESTNODE's successors. Compute f'(SUCCESSOR) = g(SUCCESSOR) + h'(SUCCESSOR)

A Search - Example*

EXAMPLE: Use A* search algorithm to find the solution.

Initial state: S, Goal state: G_1 or G_2

node	h(n)	node	h(n)	node	h(n)
A	11	D	8	H	7
B	5	E	4	I	3
C	9	F	2		

Solution:

f(A)=h(A)+g(A)=11+4=15

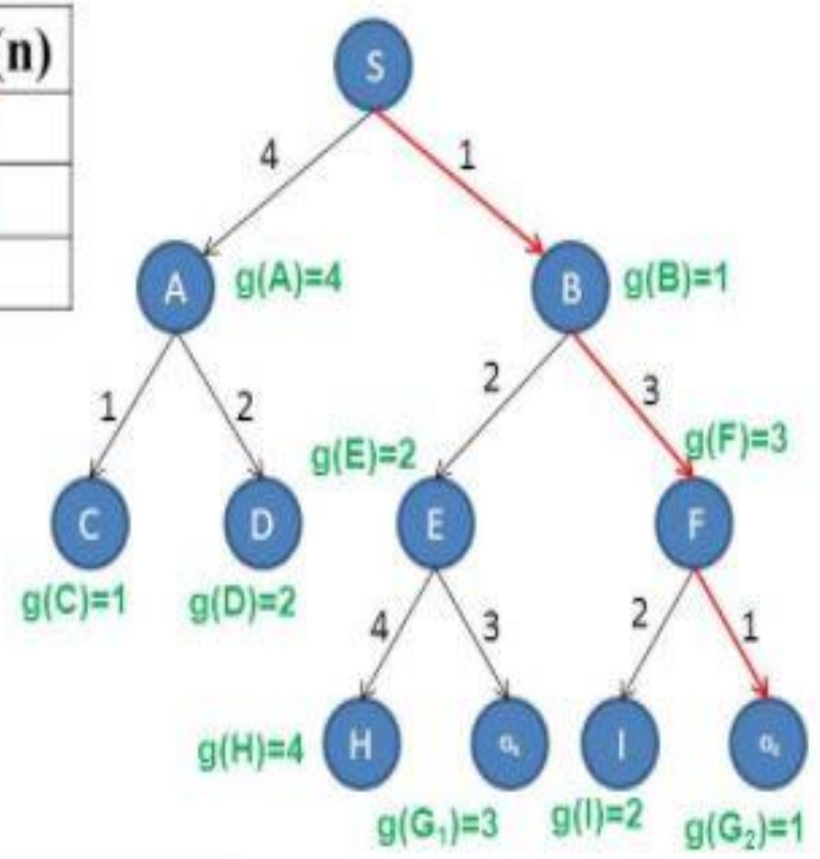
f(B)=h(B)+g(B)=5+1=6

f(E)=h(E)+g(E)=4+2=6

f(F)=h(F)+g(F)=2+3=5

f(I)=h(I)+g(I)=3+2=5

$f(G_2)=h(G_2)+g(G_2)=0+1=1$



Memory Bounded Heuristic Search

- **To Overcome space requirements, this search comes in to action.**

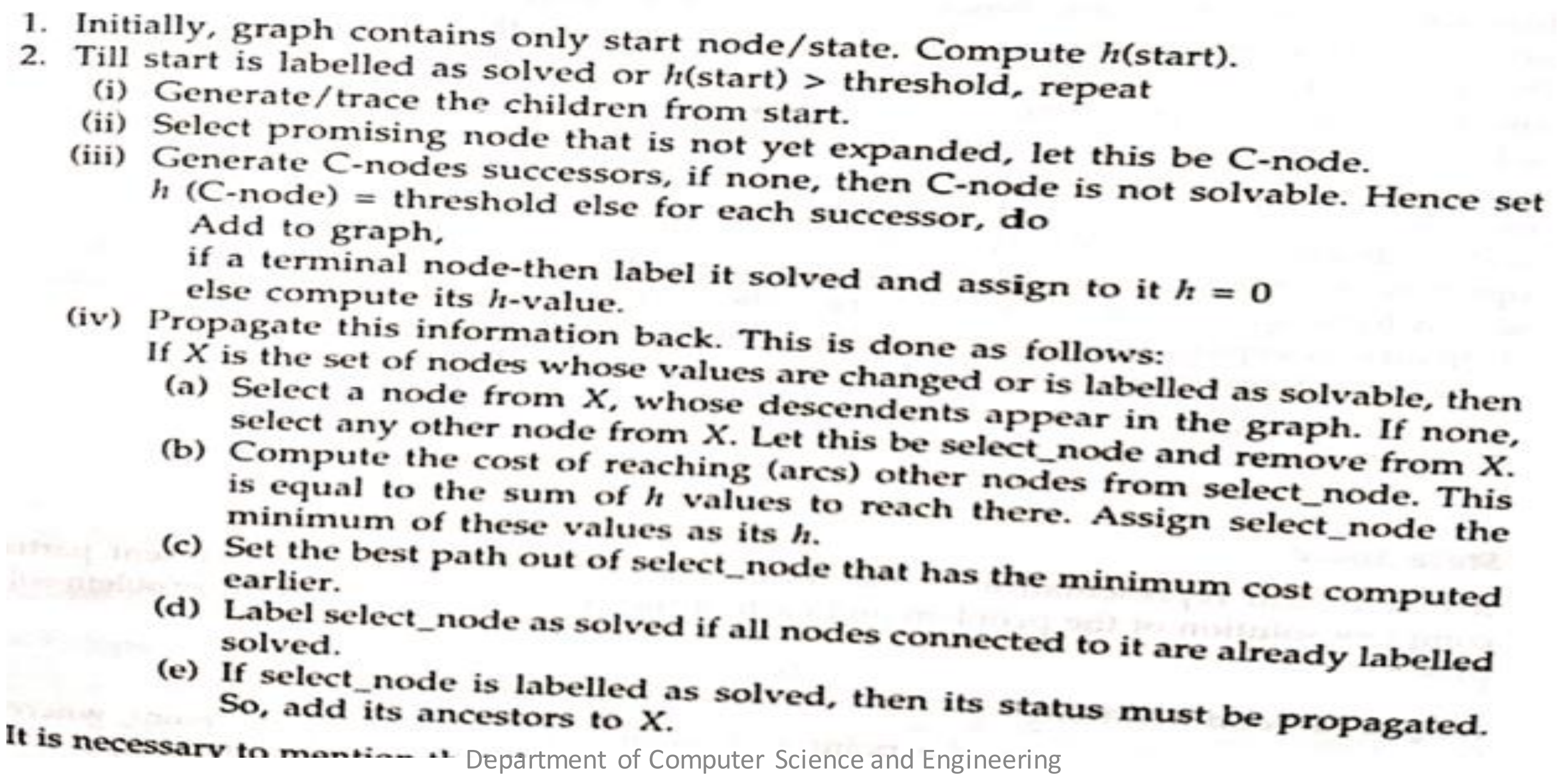
- **Iterative Deepening A* (IDA*)**
  - Similar to Iterative Deepening Search, but cut off at (g(n)+h(n)) > max instead of depth > max
  - At each iteration, cutoff is the first f-cost that exceeds the cost of the node at the previous iteration

- **RBFS**
  - It attempts to mimic the operation of BFS.
  - It replaces the f-value of each node along the path with the best f-value of its children.
  - Suffers from using too little memory.

- **Simple Memory Bounded A* (SMA*)**
  - Set max to some memory bound.
  - If the memory is full, to add a node drop the worst (g+h) node that is already stored.
  - Expands newest best leaf, deletes oldest worst leaf.

AO* Search

- A* algorithm uses open and closed list to maintain the node status.
- AO* instead maintains the entire graph that has been generated till the current state.
- So, every node or state stores its h-value rather than storing the g-value



```
1. Initially, graph contains only start node/state. Compute h(start).
2. Till start is labelled as solved or h(start) > threshold, repeat
   (i)   Generate/trace the children from start.
   (ii)  Select promising node that is not yet expanded, let this be C-node.
   (iii) Generate C-nodes successors, if none, then C-node is not solvable. Hence set
         h (C-node) = threshold else for each successor, do
            Add to graph,
            if a terminal node-then label it solved and assign to it h = 0
            else compute its h-value.
   (iv)  Propagate this information back. This is done as follows:
         If X is the set of nodes whose values are changed or is labelled as solvable, then
         (a) Select a node from X, whose descendents appear in the graph. If none,
             select any other node from X. Let this be select_node and remove from X.
         (b) Compute the cost of reaching (arcs) other nodes from select_node. This
             is equal to the sum of h values to reach there. Assign select_node the
             minimum of these values as its h.
         (c) Set the best path out of select_node that has the minimum cost computed
             earlier.
         (d) Label select_node as solved if all nodes connected to it are already labelled
             solved.
         (e) If select_node is labelled as solved, then its status must be propagated.
             So, add its ancestors to X.
It is necessary to mention
```

Local search Algorithms

- **Local search** is a heuristic method for solving computationally hard optimization problems.
- Local search can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions.
- Local search algorithms move from solution to solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed.
- Here, each step is a typical transition to one of its neighbourhood from present state.
- The move is based on value of cost function.

- Two important parameters to be considered are selected initial solution and the stop criterion.
- The stop criterion typically defines the condition when the search phase is over and the best solution found is returned.
- The three main entities are
  - Search Space
  - Neighbourhood Relations
  - Cost Function

- **Advantages**: Very little memory — usually constant amount- Can often find reasonable solutions in infinite (continuous) state spaces
- **Nutshell:**
  - All that matters is the solution state
  - Don't care about solution path
- **Examples**
  - Hill Climbing Search
  - Simulated Annealing (suited for either local or global search)
    - Non Traditional Search and Optimization technique
  - Genetic Algorithm
    - Non Traditional Search and Optimization technique
    - Conceived by Prof. Holland of University of Michigan

Hill Climbing Search

- 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.

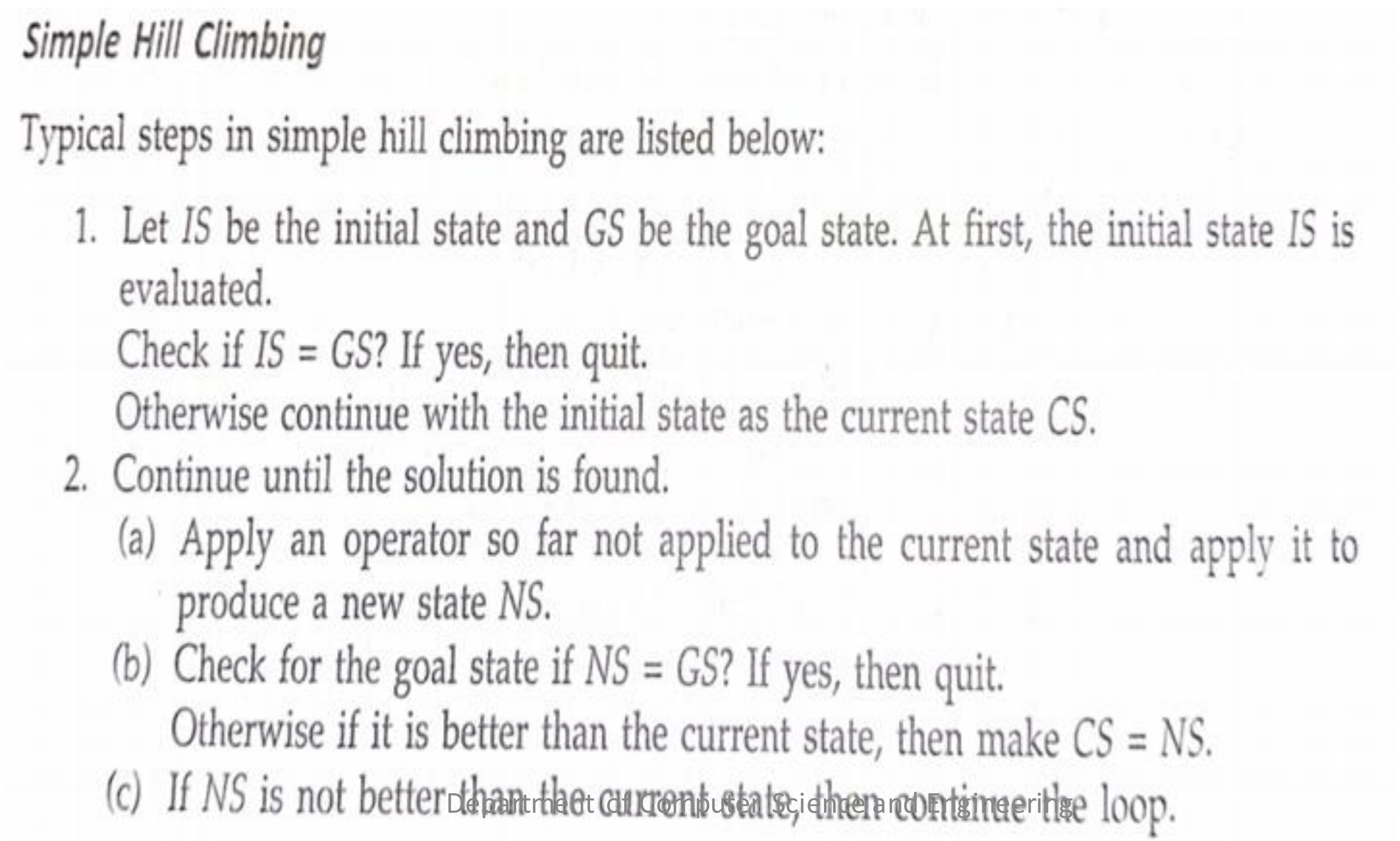
Features of Hill Climbing:

- **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.

Types of Hill Climbing

• Simple hill climbing :

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

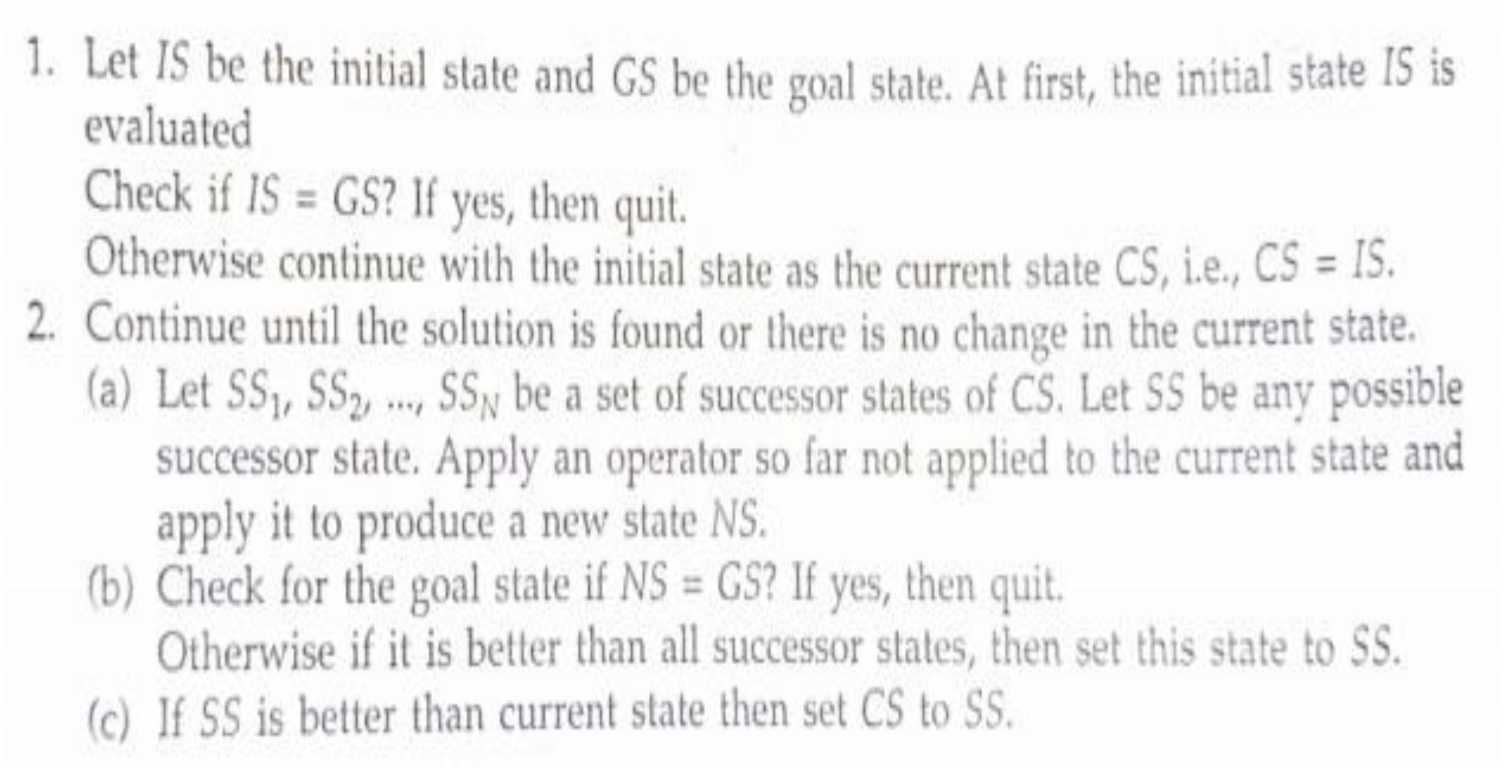


Simple Hill Climbing

Typical steps in simple hill climbing are listed below:

1. Let *IS* be the initial state and *GS* be the goal state. At first, the initial state *IS* is evaluated.
   Check if *IS* = *GS*? If yes, then quit.
   Otherwise continue with the initial state as the current state CS.
2. Continue until the solution is found.
   (a) Apply an operator so far not applied to the current state and apply it to produce a new state *NS*.
   (b) Check for the goal state if *NS* = *GS*? If yes, then quit.
   Otherwise if it is better than the current state, then make CS = *NS*.
   (c) If *NS* is not better than the current state, then continue the loop.

- **Steepest-Ascent Hill climbing:**
  - It first examines all the neighboring nodes and then selects the node closest to the solution state as of next node.
  - The algorithm works as follows:



1. Let *IS* be the initial state and *GS* be the goal state. At first, the initial state *IS* is evaluated
   Check if *IS* = *GS*? If yes, then quit.
   Otherwise continue with the initial state as the current state *CS*, i.e., *CS* = *IS*.
2. Continue until the solution is found or there is no change in the current state.
   (a) Let SS_1, SS_2, ..., SS_N be a set of successor states of *CS*. Let *SS* be any possible successor state. Apply an operator so far not applied to the current state and apply it to produce a new state *NS*.
   (b) Check for the goal state if *NS* = *GS*? If yes, then quit.
   Otherwise if it is better than all successor states, then set this state to *SS*.
   (c) If *SS* is better than current state then set *CS* to *SS*.

- **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.
- **First Choice hill climbing:**
  - It is a varient of stochastic hill climbing.
  - In this algorithm, the successor is generated randomly untill the one generated is better than the current state.
  - when there are large no of successor states then this is a better option.

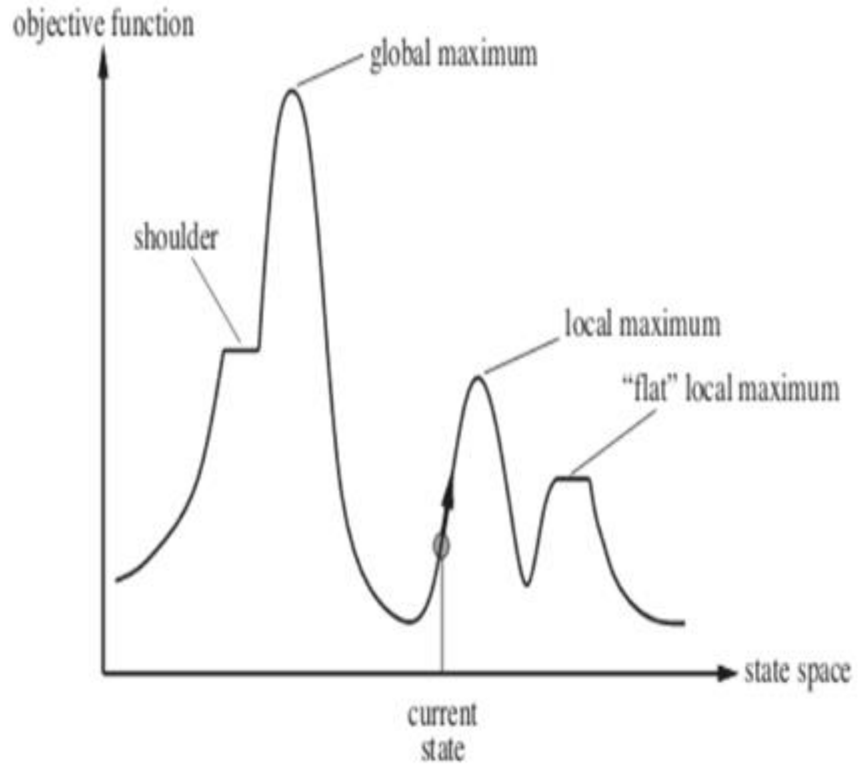
State Space diagram for Hill Climbing

- 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 that state space where objective function has 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 because at this state, objective function has highest value.
- **Plateua/flat local maximum :** It is a flat region of state space where neighboring states have the same value.
- **Ridge :** It is region which is higher than its neighbours but itself has a slope. It is a special kind of local maximum.
- **Current state :** The region of state space diagram where we are currently present during the search.
- **Shoulder :** It is a plateau that has an uphill edge.

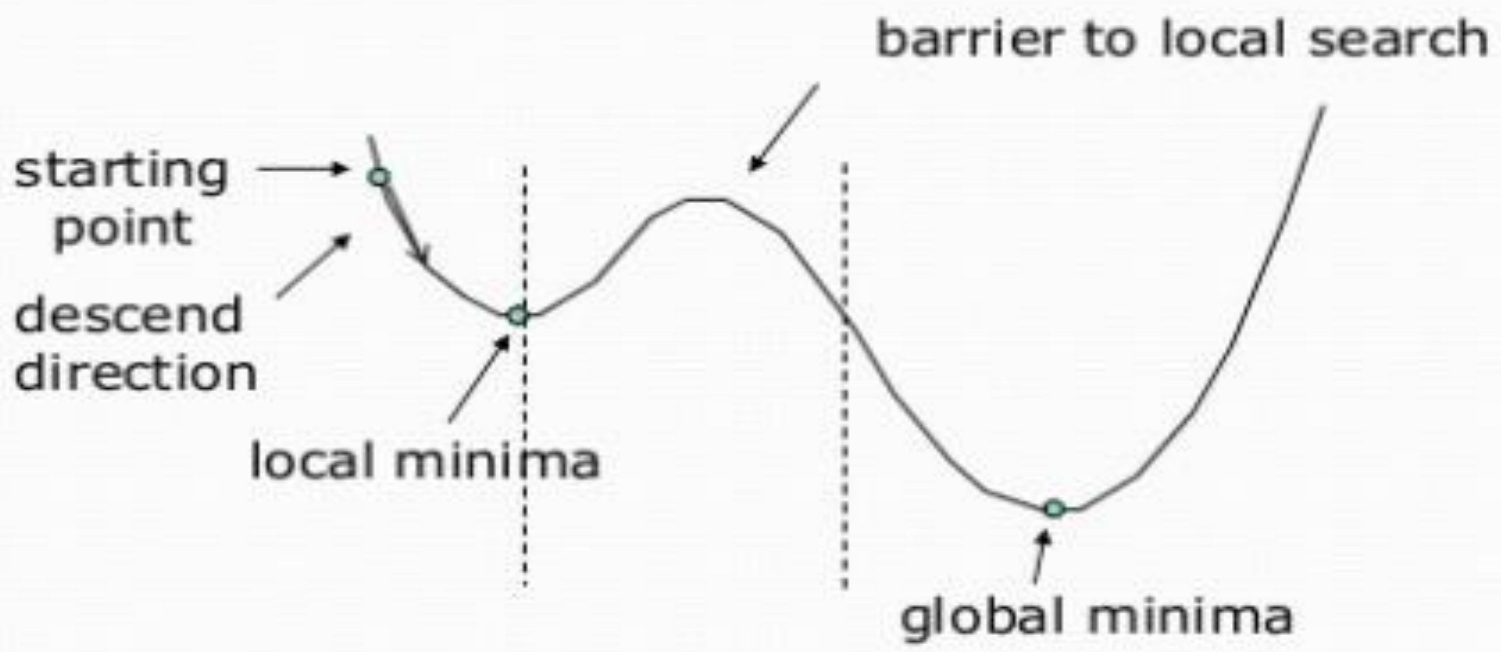
Problems in different regions in Hill climbing

- Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :
  - **Local maximum :** At a local maximum all neighboring states have a values which is worse than the current state. Since hill-climbing uses a greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.
    - **To overcome local maximum problem :** Utilize backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.
  - **Plateau :** On plateau all neighbors have same value . Hence, it is not possible to select the best direction.
    - **To overcome plateaus :** Make a big jump. Randomly select a state far away from the current state. Chances are that we will land at a non-plateau region
  - **Ridge :** Any point on a ridge can look like peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state.
    - **To overcome Ridge :** In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

Simulated Annealing

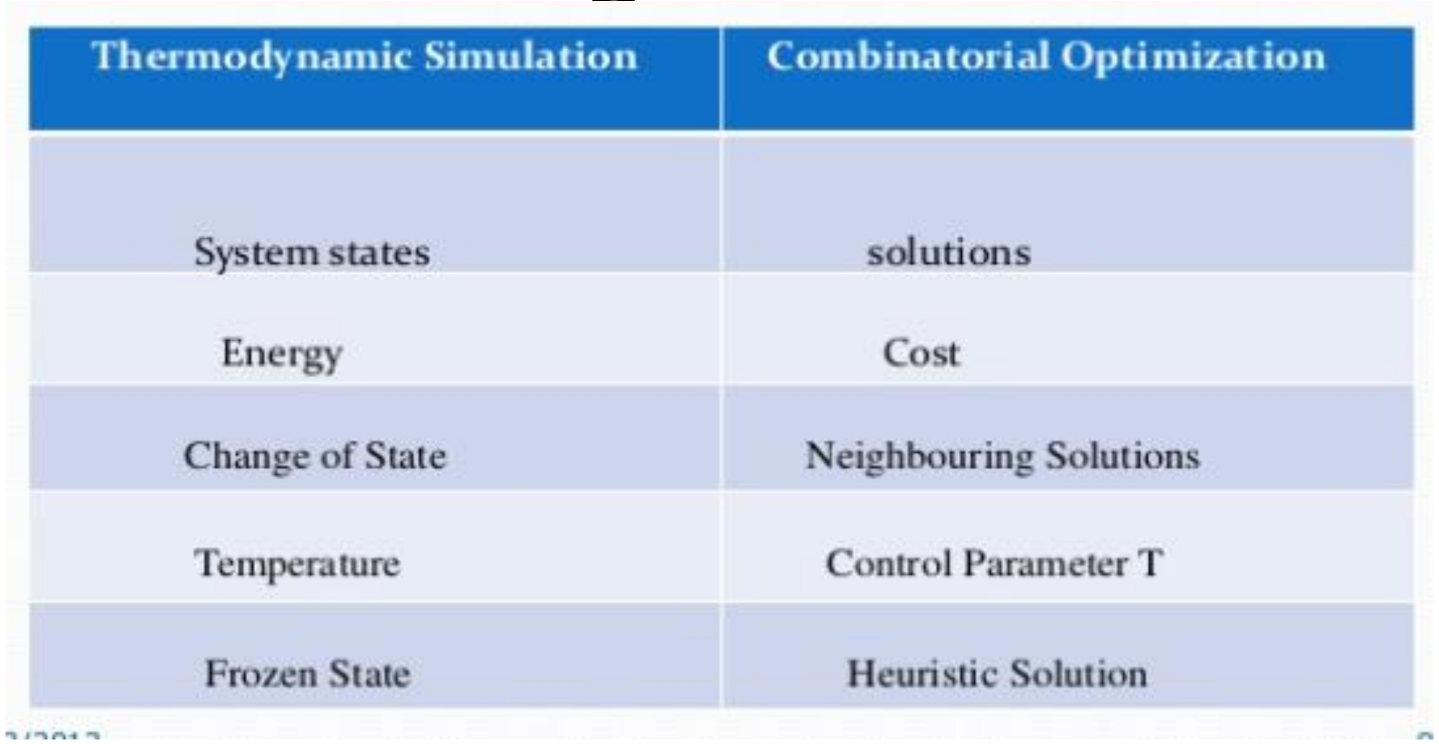
- *Simulated Annealing came from the concept of annealing in physics. This technique is used to increase the size of crystals and to reduce the defects in crystals. This was done by heating and then suddenly cooling of crystals.*
- *Advantages*
  - can deal with arbitrary systems and cost functions
  - is relatively easy to code, even for complex problems
  - generally gives a "good" solution

Difficulty in Searching Global Optima



- SA is a global optimization technique.
- SA distinguishes between different local optima.
- SA is a memory less algorithm, the algorithm does not use any information gathered during the search ⍰
- SA is motivated by an analogy to annealing in solids. ⍰
- Simulated Annealing – an iterative improvement algorithm.
- Simulated annealing establishes the connection between this type of thermodynamic behaviour and the search for global minima for a discrete optimization problem.

Relationship between Physical Annealing and Simulated Annealing



Thermodynamic Simulation	Combinatorial Optimization
System states	solutions
Energy	Cost
Change of State	Neighbouring Solutions
Temperature	Control Parameter T
Frozen State	Heuristic Solution

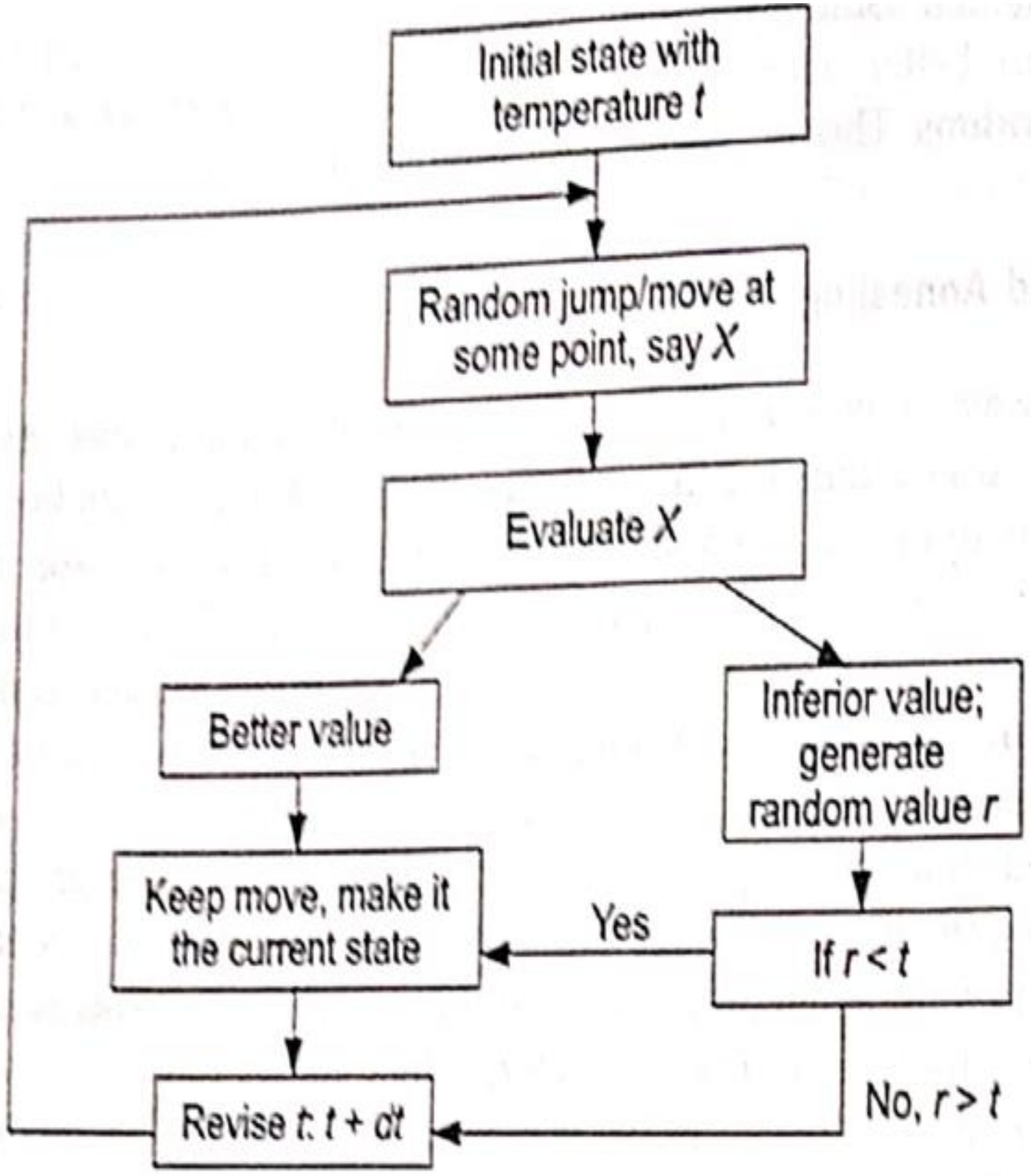
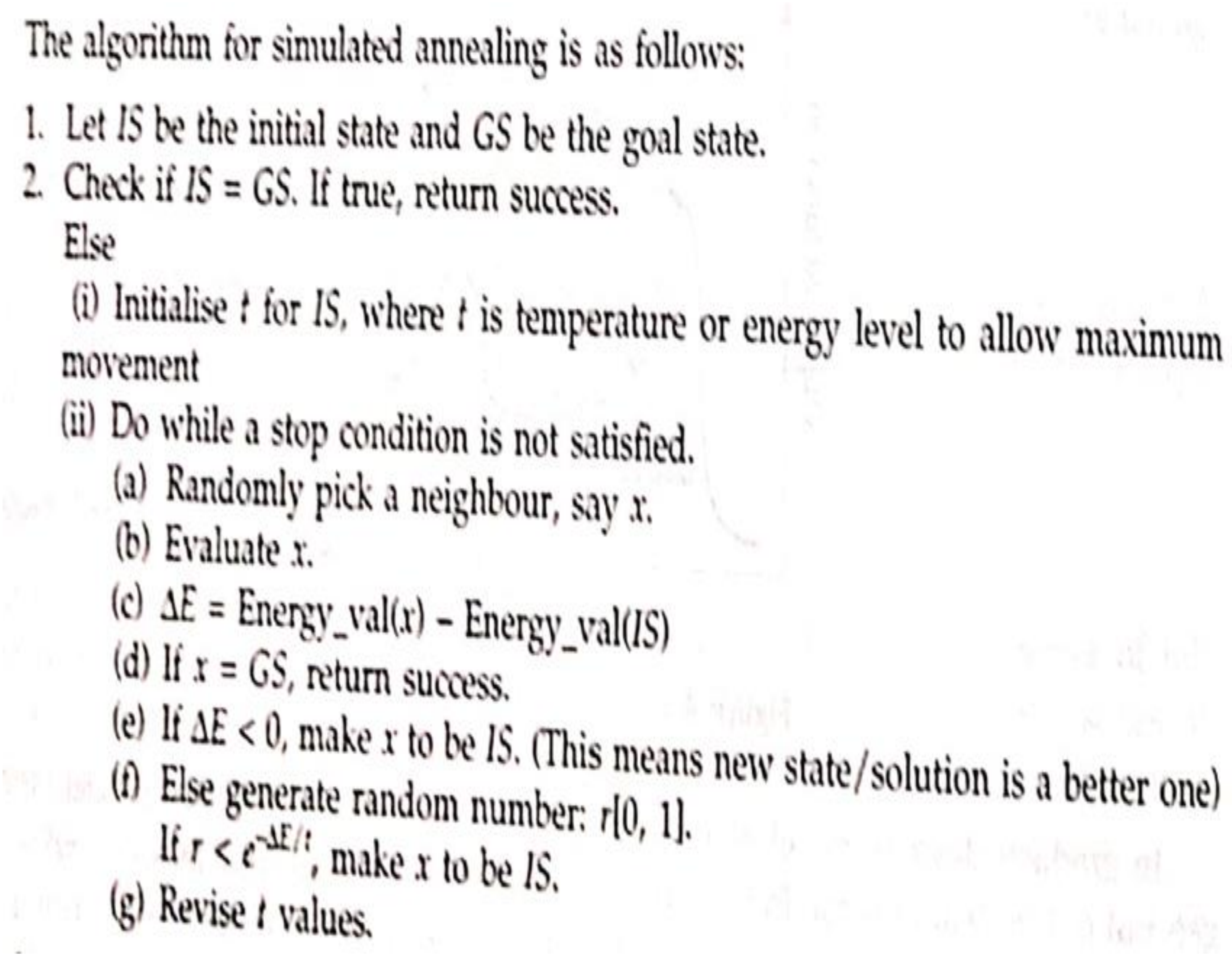
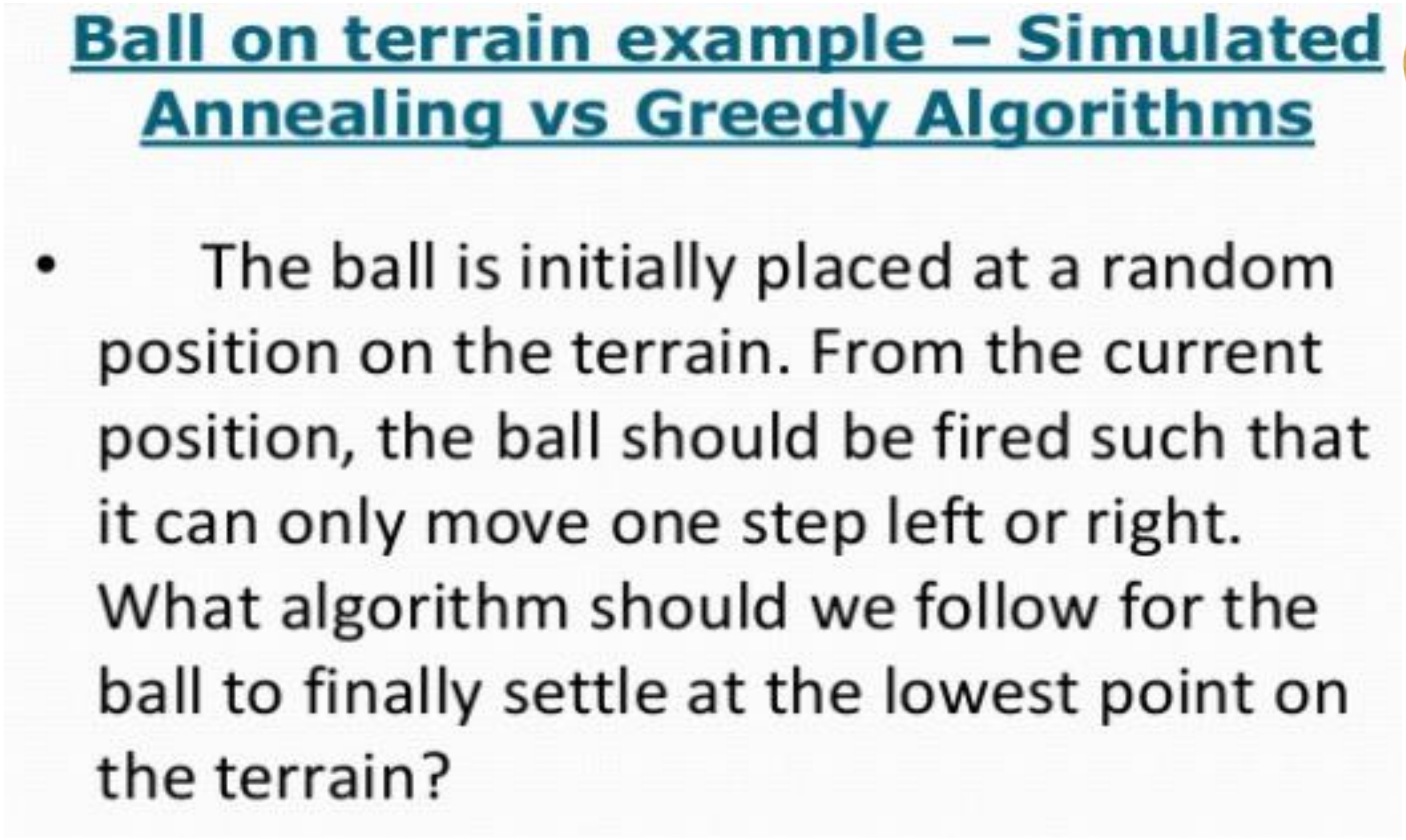

Initial state with temperature t
Random jump/move at some point, say X
Evaluate X
Better value
Inferior value; generate random value r
Keep move, make it the current state
Yes
If $r < t$
No, $r > t$
Revise t: $t + dt$

Figure 4.11 Simulated annealing: Basic steps.



The algorithm for simulated annealing is as follows:

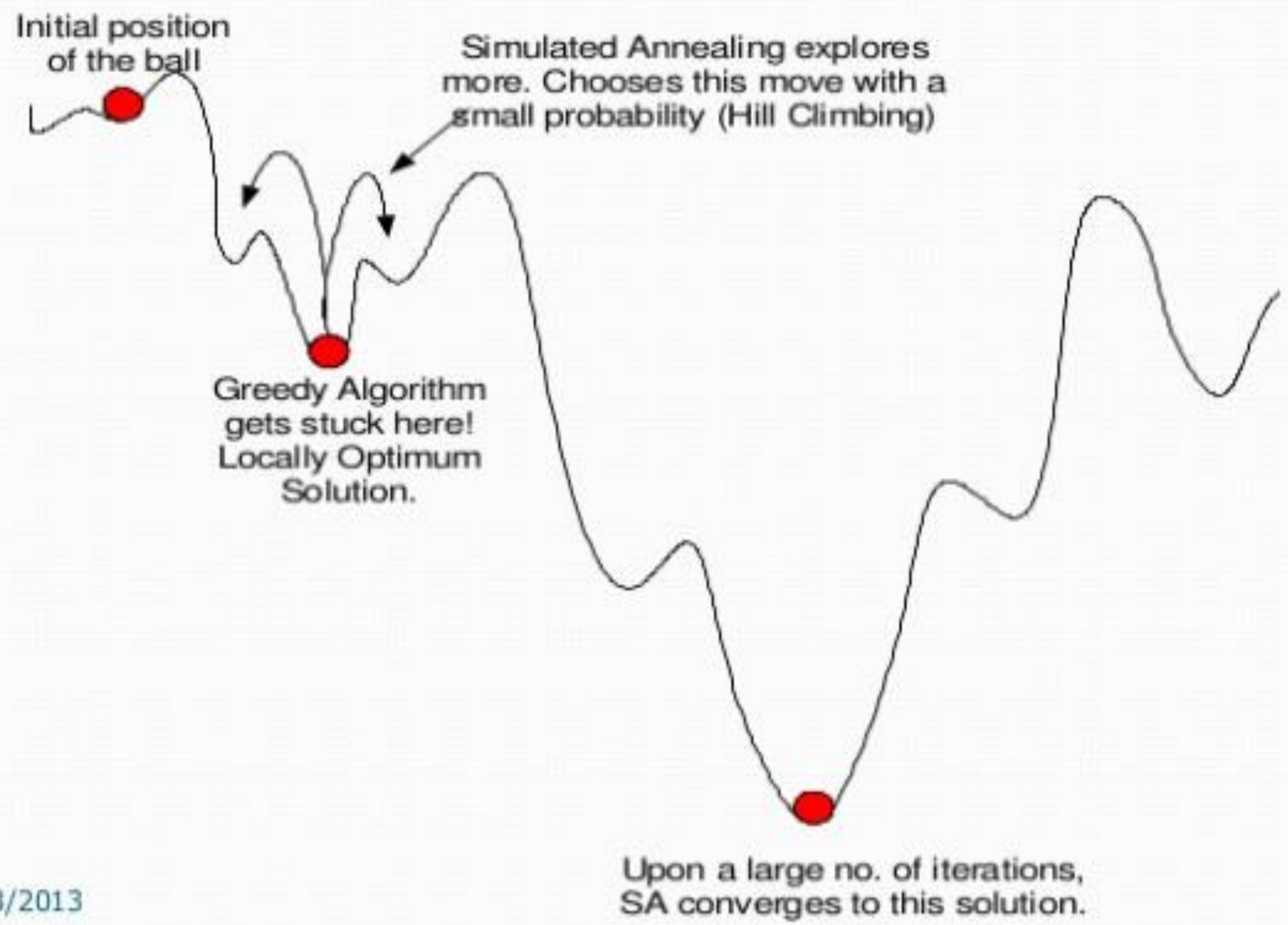
```
1. Let $IS$ be the initial state and $GS$ be the goal state.
2. Check if $IS = GS$. If true, return success.
   Else
    (i) Initialise $t$ for $IS$, where $t$ is temperature or energy level to allow maximum
   movement
   (ii) Do while a stop condition is not satisfied.
        (a) Randomly pick a neighbour, say $x$.
        (b) Evaluate $x$.
        (c) $\Delta E$ = Energy_val($x$) – Energy_val($IS$)
        (d) If $x = GS$, return success.
        (e) If $\Delta E < 0$, make $x$ to be $IS$. (This means new state/solution is a better one)
        (f) Else generate random number: $r[0, 1]$.
            If $r < e^{-\Delta E/t}$, make $x$ to be $IS$.
        (g) Revise $t$ values.
```



Ball on terrain example – Simulated Annealing vs Greedy Algorithms

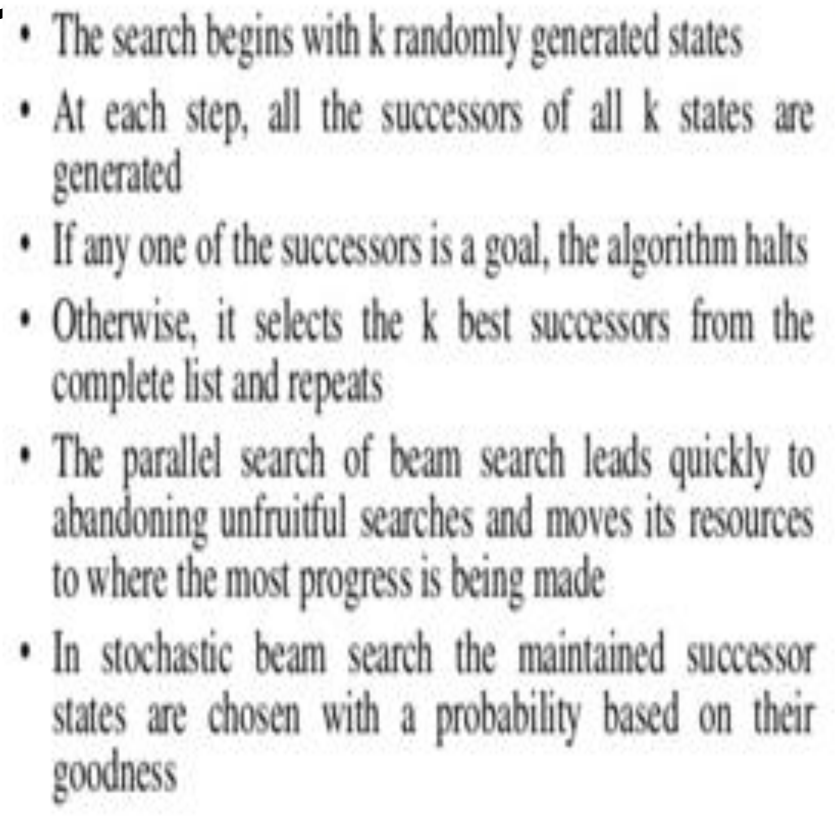
- The ball is initially placed at a random position on the terrain. From the current position, the ball should be fired such that it can only move one step left or right. What algorithm should we follow for the ball to finally settle at the lowest point on the terrain?

Ball on terrain example – SA vs. Greedy Algorithms

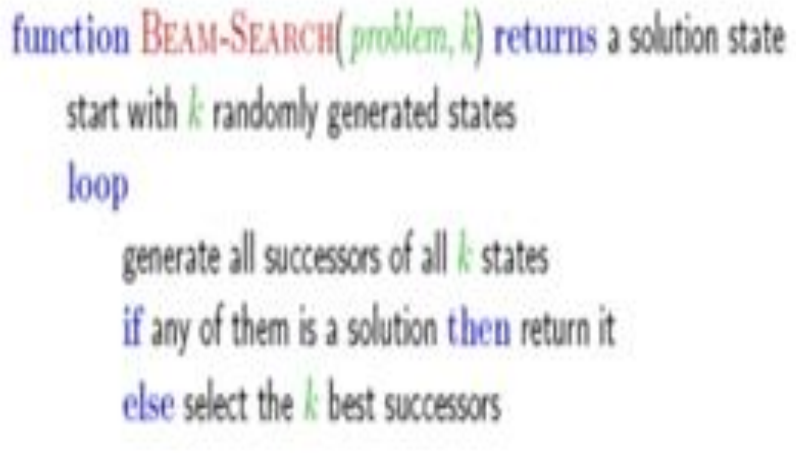

Initial position of the ball
Simulated Annealing explores more. Chooses this move with a small probability (Hill Climbing)
Greedy Algorithm gets stuck here! Locally Optimum Solution.
Upon a large no. of iterations, SA converges to this solution.

23/2013

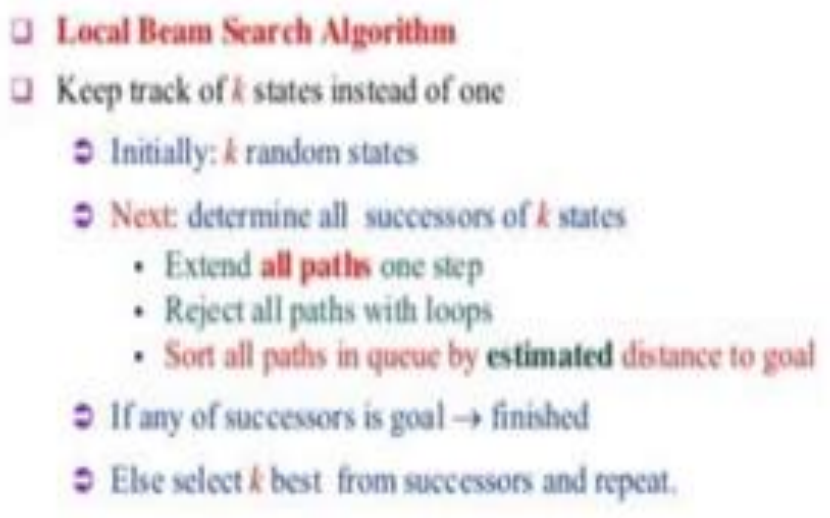
Local Beam Search



- The search begins with k randomly generated states
- At each step, all the successors of all k states are generated
- If any one of the successors is a goal, the algorithm halts
- Otherwise, it selects the k best successors from the complete list and repeats
- The parallel search of beam search leads quickly to abandoning unfruitful searches and moves its resources to where the most progress is being made
- In stochastic beam search the maintained successor states are chosen with a probability based on their goodness



```
function BEAM-SEARCH(problem, k) returns a solution state
    start with k randomly generated states
    loop
        generate all successors of all k states
        if any of them is a solution then return it
        else select the k best successors
```



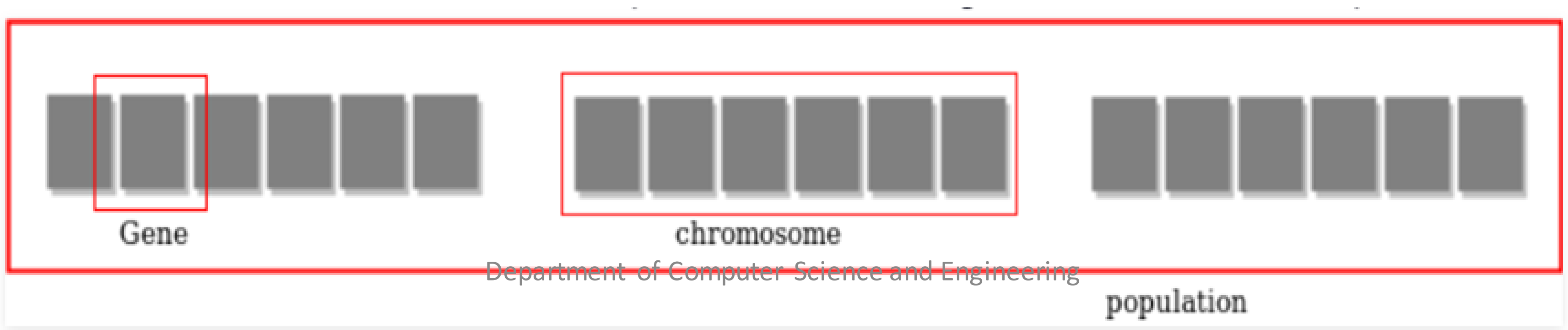
- **Local Beam Search Algorithm**
- Keep track of *k* states instead of one
  - Initially: *k* random states
  - Next: determine all successors of *k* states
    - Extend **all paths** one step
    - Reject all paths with loops
    - Sort all paths in queue by **estimated** distance to goal
  - If any of successors is goal → finished
  - Else select *k* best from successors and repeat.

Genetic Algorithms

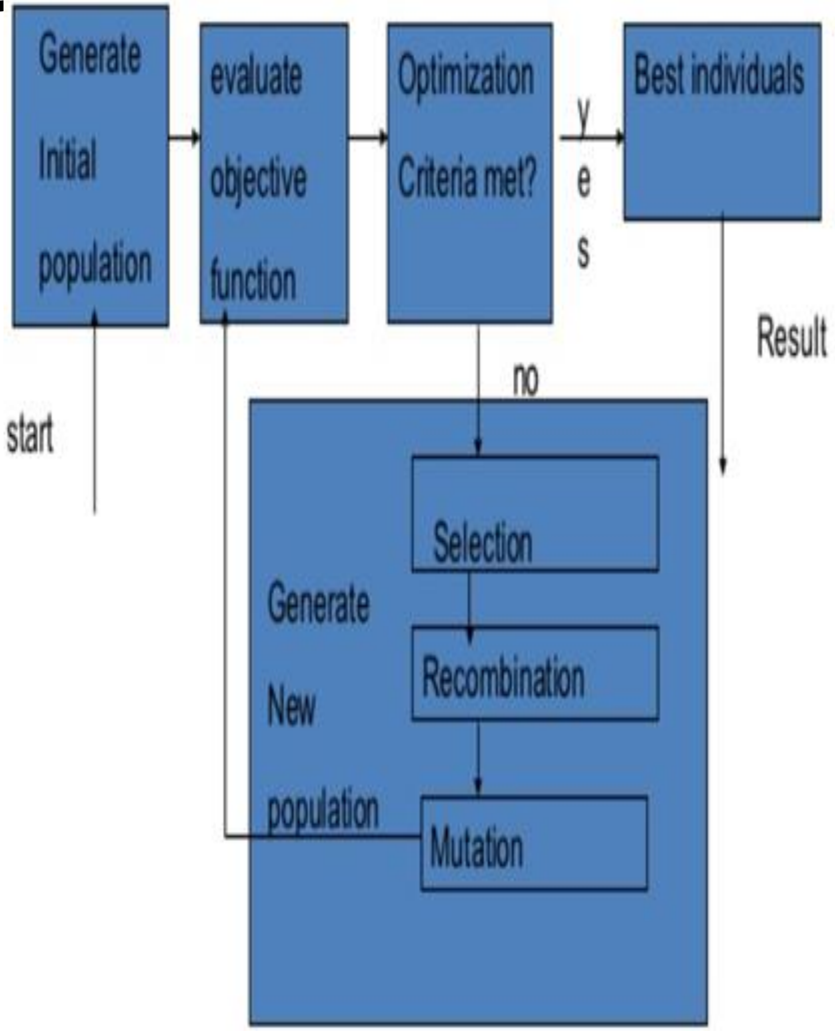
- Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics.
- **They are commonly used to generate high-quality solutions for optimization problems and search problems.**
- In simple words, they simulate "survival of the fittest" among individual of consecutive generation for solving a problem. **Each generation consist of a population of individuals** and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome

Search Space

- The population of individuals are maintained within search space. Each individual represent a solution in search space for given problem. Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes. Thus a chromosome (individual) is composed of several genes (variable components).
- A Fitness Score is given to each individual which **shows the ability of an individual to "compete"**. The individual having optimal fitness score (or near optimal) are sought.



Genetic Algorithms



- Genetic algorithms are based on the theory of selection

  1.A set of random solutions are generated
- Only those solutions survive that satisfy a fitness function
- Each solution in the set is a chromosome
- A set of such solutions forms a population

2. The algorithm uses three basic genetic operators namely

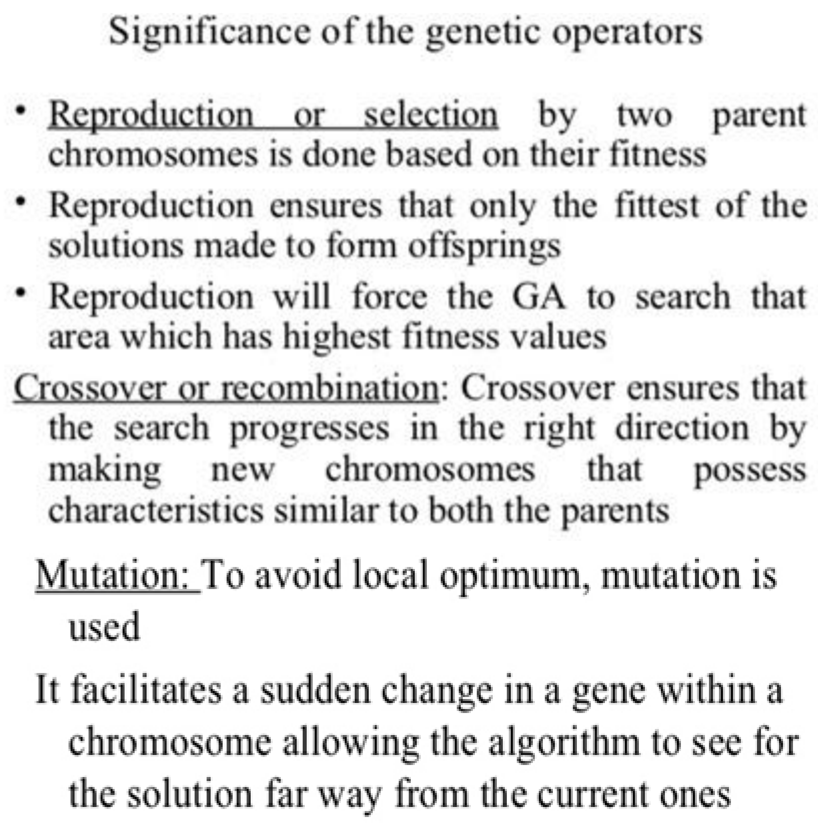
(i)Reproduction

(ii)crossover and

(iii) mutation along with a fitness function to evolve a new population or the next generation

- Thus the algorithm uses these operators and the fitness function to guide its search for the optimal solution
- It is a guided random search mechanism

S8-Genetic Algorithms



Significance of the genetic operators

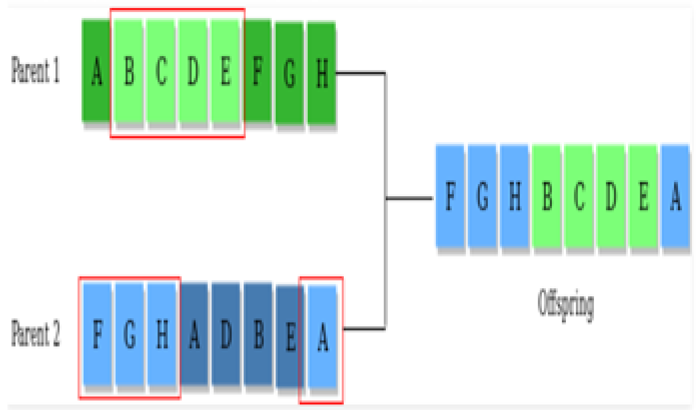
- <u>Reproduction or selection</u> by two parent chromosomes is done based on their fitness
- Reproduction ensures that only the fittest of the solutions made to form offsprings
- Reproduction will force the GA to search that area which has highest fitness values

<u>Crossover or recombination</u>: Crossover ensures that the search progresses in the right direction by making new chromosomes that possess characteristics similar to both the parents

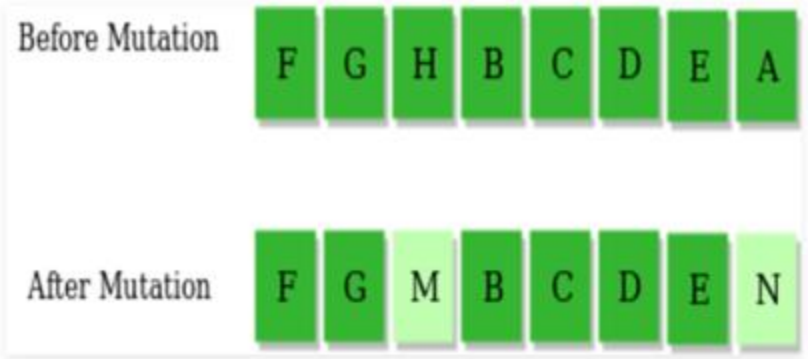
<u>Mutation:</u> To avoid local optimum, mutation is used

It facilitates a sudden change in a gene within a chromosome allowing the algorithm to see for the solution far way from the current ones

- Cross Over



- Mutation



GA can be summarized as:

1) Randomly initialize populations p
2) Determine fitness of population
3) Untill convergence repeat:
   - a) Select parents from population
   - b) Crossover and generate new population
   - c) Perform mutation on new population
   - d) Calculate fitness for new population

Adversarial search Methods-Game playing

- Adversarial search: Search based on Game theory- Agents- Competitive environment
- **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games.**
- ***According to game theory, a game is played between two players. To complete the game, one has to win the game and the other looses automatically.'***
- Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

Adversarial search Methods-Game playing-Important concepts

My Interest
"I Win"
Your Interest
"You Loose"

We are opponents- I win, you loose.

- **Techniques required to get the best optimal solution (Choose Algorithms for best optimal solution within limited time)**
  - **Pruning:** A technique which allows ignoring the unwanted portions of a search tree which make no difference in its final result.
  - **Heuristic Evaluation Function:** It allows to approximate the cost value at each level of the search tree, before reaching the goal node.

Adversarial Search

Types of Games in AI:

	Deterministic	Chance Moves
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

Game playing-Important concepts

- Some of the important concepts related to games are game classes, game strategies and game equilibriums.

- **Game Classes:** Diversified approach has resulted in several classes of games.
  - symmetric game
  - zero-sum game
  - perfect information game
  - simultaneous game

- **Game Strategies:**
- Move is an action taken by a player at some point during the progress of a game.
- Strategy is a complete approach for playing the game.
  - Dominant strategy
  - Pure strategy
  - Mixed strategy
  - Tit for tat
  - Collusion
  - Backward induction

- **Game Equilibrium :**
- Game theory attempts to find the state of equilibrium.
- In equilibrium, each player of the game adopts a strategy.
- Equilibrium depends on the field of the application, although it may coincide.
- Equilibrium is about selection of a stable state.

Game as a search problem

- **Types of algorithms in Adversarial search**
  - In a **normal search**, we follow a sequence of actions to reach the goal or to finish the game optimally.
  - But in an **adversarial search**, the result depends on the players which will decide the result of the game.
  - It is also obvious that the solution for the goal state will be an optimal solution because the player will try to win the game with the shortest path and under limited time.
    - **Minmax Algorithm**
    - **Alpha-beta Pruning**

Mini-Max Algorithm in Artificial Intelligence

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory.
- It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- **Min-Max algorithm is mostly used for game playing in AI**. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.

Mini-Max Algorithm in Artificial Intelligence

- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Pseudo-code for MinMax Algorithm

```
function minimax(node, depth, maximizingPlayer) is
if depth ==0 or node is a terminal node then
return static evaluation of node

if MaximizingPlayer then      // for Maximizer Player
maxEva= -infinity
 for each child of node do
 eva= minimax(child, depth-1, false)
maxEva= max(maxEva,eva)      //gives Maximum of t
he values
return maxEva
```

```
else          // for Minimizer player
 minEva= +infinity
 for each child of node do
 eva= minimax(child, depth-1, true)
 minEva= min(minEva, eva)      //gi
ves minimum of the values
 return minEva
```

Mini-Max Algorithm in Artificial Intelligence

Working of Min-Max Algorithm:

- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs.

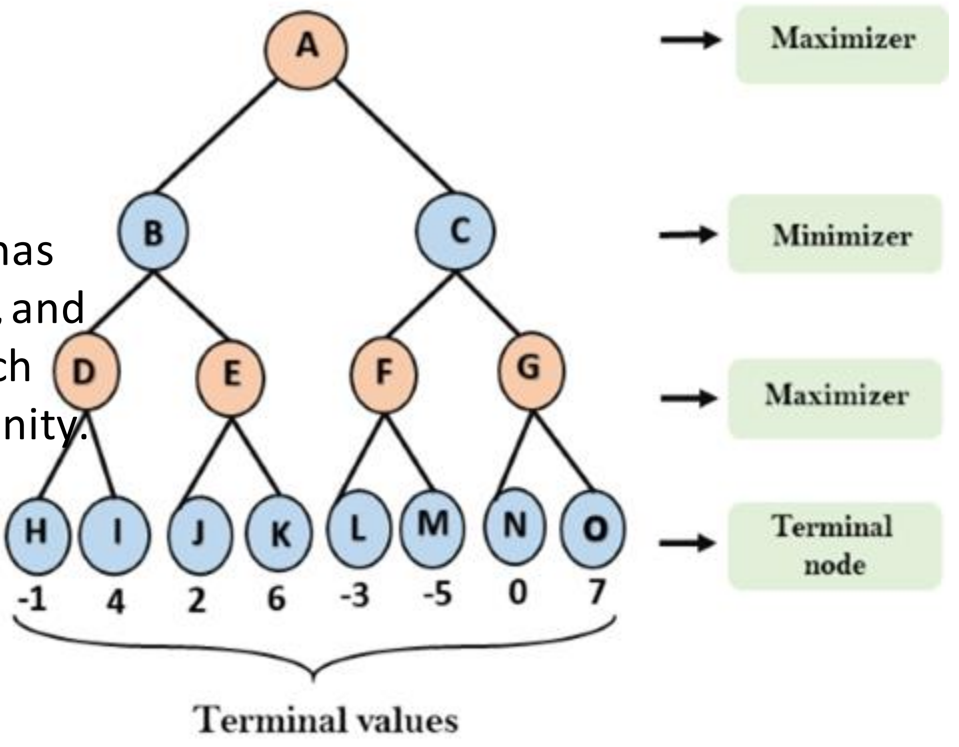
Mini-Max Algorithm in Artificial Intelligence

Working of Min-Max Algorithm:

- Following are the main steps involved in solving the two-player game tree:

- **Step-1:**

Maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



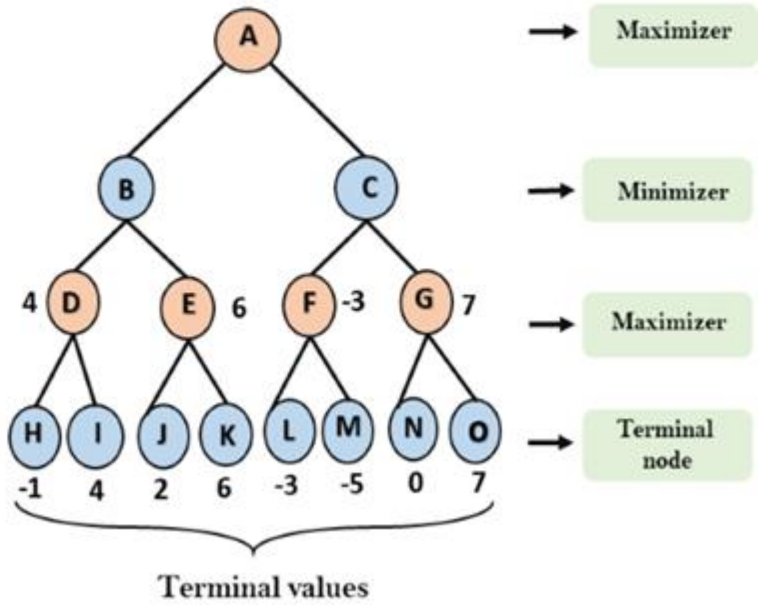
Mini-Max Algorithm in Artificial Intelligence

Working of Min-Max Algorithm:

- Following are the main steps involved in solving the two-player game tree:

- **Step-2:**

For node D	max(-1,- -∞) => max(-1,4)= 4
For Node E	max(2, -∞) => max(2, 6)= 6
For Node F	max(-3, -∞) => max(-3,-5) = -3
For node G	max(0, -∞) = max(0, 7) = 7



Mini-Max Algorithm in Artificial Intelligence

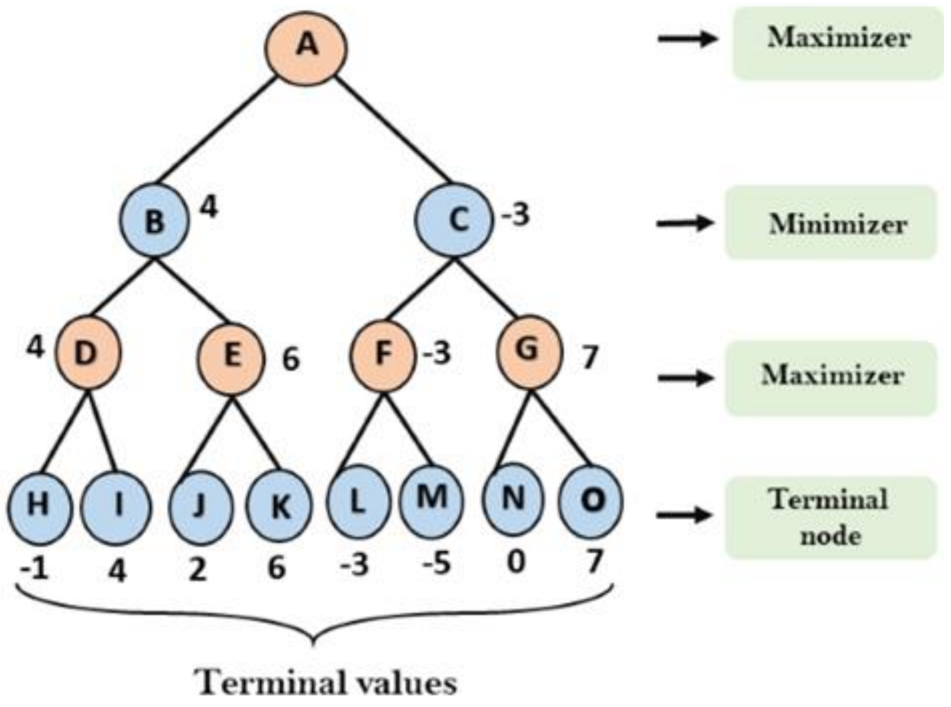
Working of Min-Max Algorithm:

- Following are the main steps involved in solving the two-player game tree:

- **Step-3:**

For node B= min(4,6) = 4
For node C= min (-3, 7) = -3

For node F= max(-3,-5)= -3
For node G= max(0,7)= 7



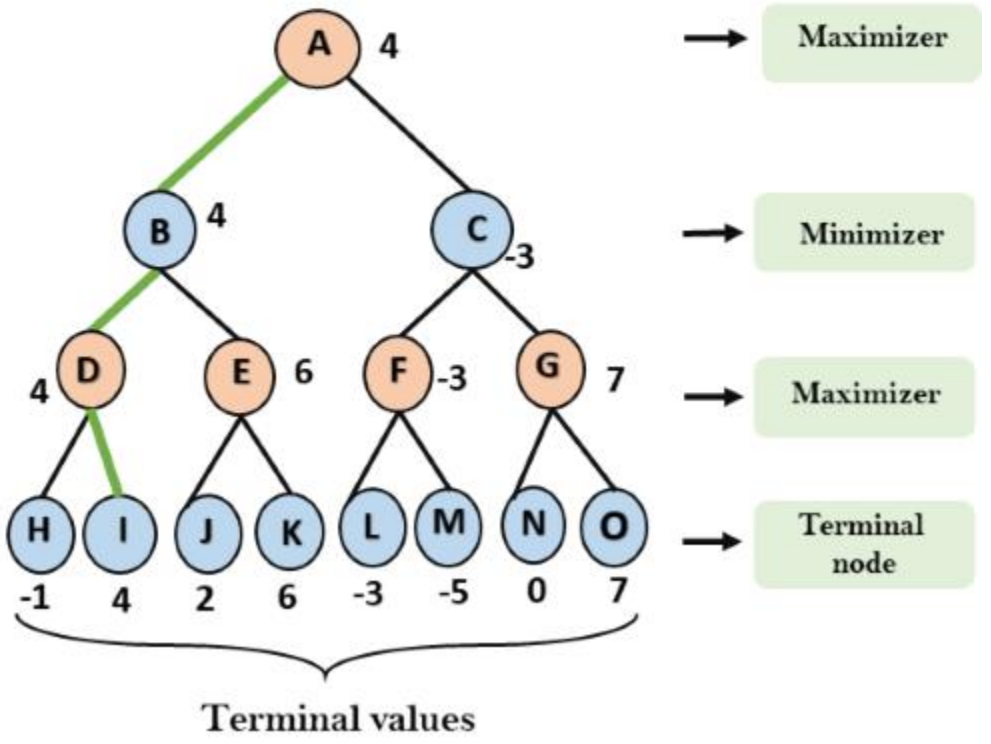
Mini-Max Algorithm in Artificial Intelligence

Working of Min-Max Algorithm:

- Following are the main steps involved in solving the two-player game tree:

- **Step-3:**

For node A max(4, -3)= 4



Alpha beta pruning

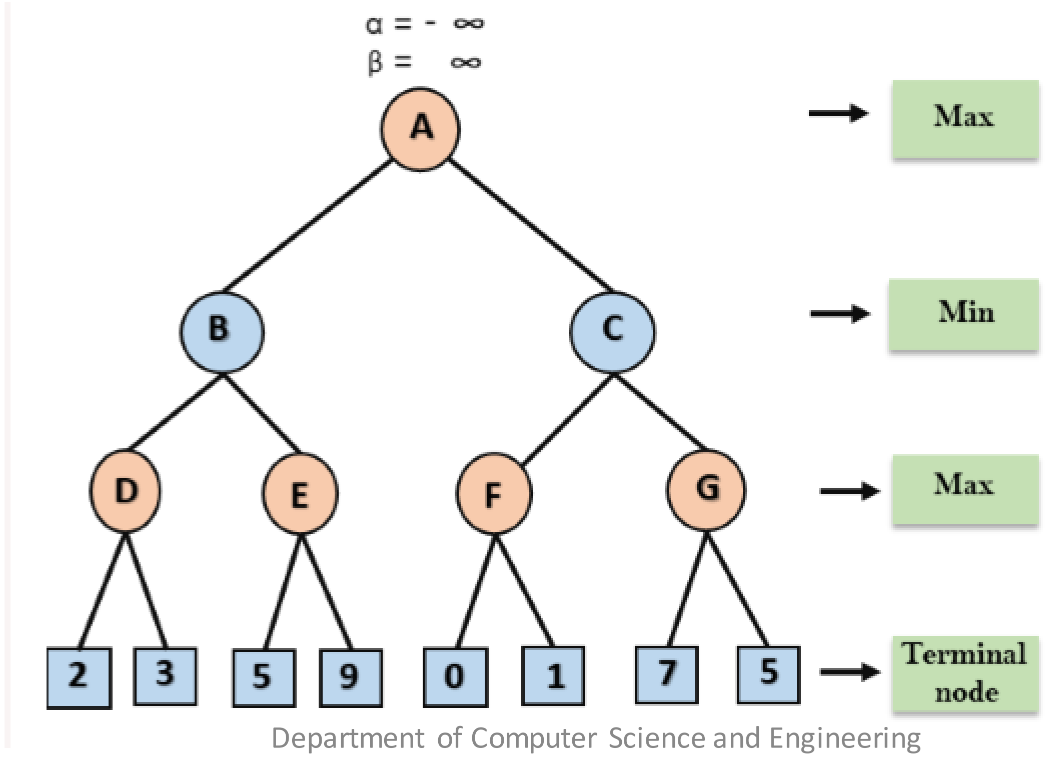
- Alpha-beta pruning is a **modified version** of the minimax algorithm. It is an **optimization technique** for the minimax algorithm.
- Drawback of Minimax:
  - Explores each node in the tree deeply to provide the best path among all the paths
  - Increases time complexity
- Alpha beta pruning: Overcomes drawback by less exploring nodes of search tree
- Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning.**

- This involves two threshold parameter **Alpha and beta** for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm.**
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
  - **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.
  - **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.

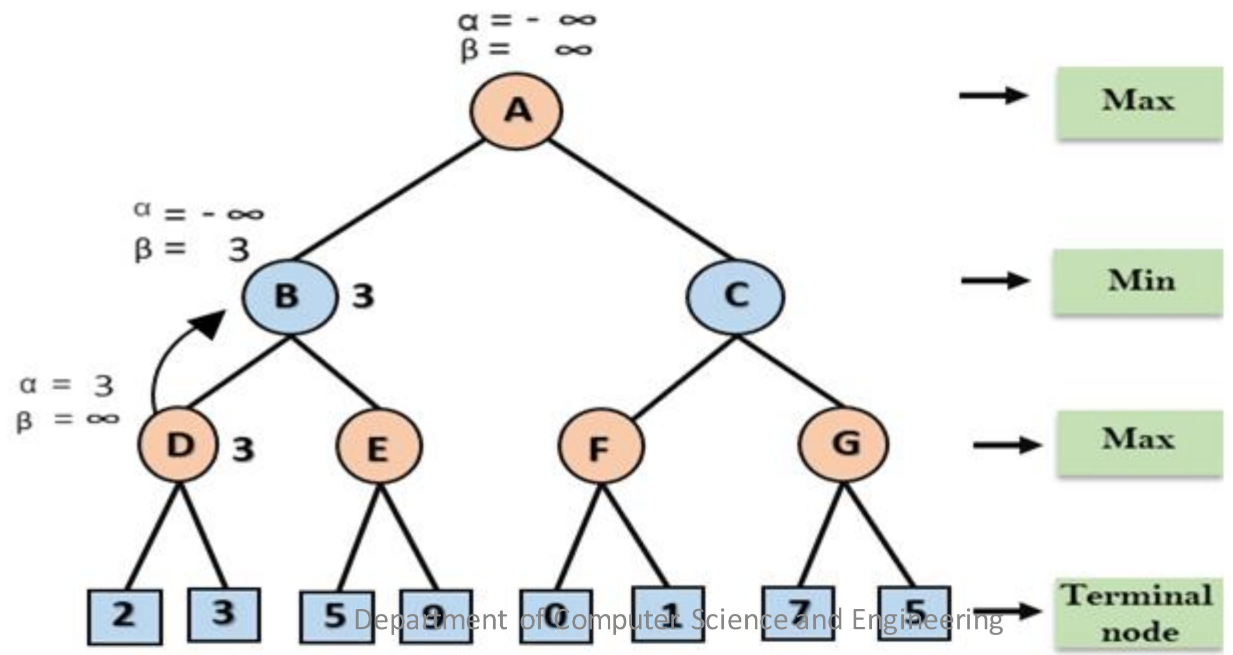
- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow.
- Hence by pruning these nodes, it makes the algorithm fast.
- The main condition which required for alpha-beta pruning is: $\alpha >= \beta$
- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta. We will only pass the alpha, beta values to the child nodes.

Working of Alpha-Beta Pruning:

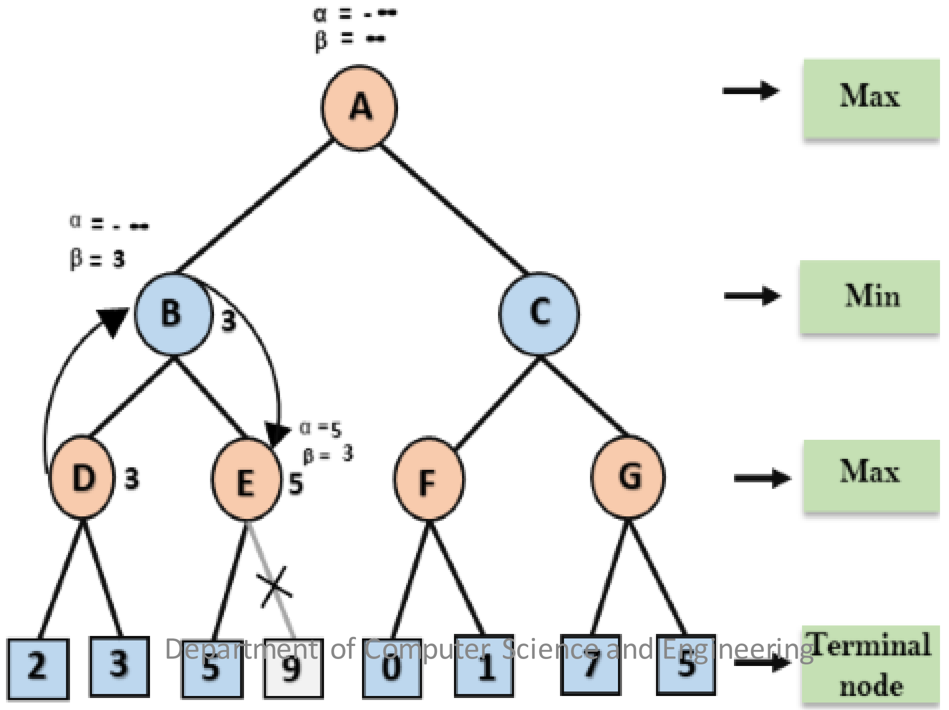
Step 1: At the first step, the Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



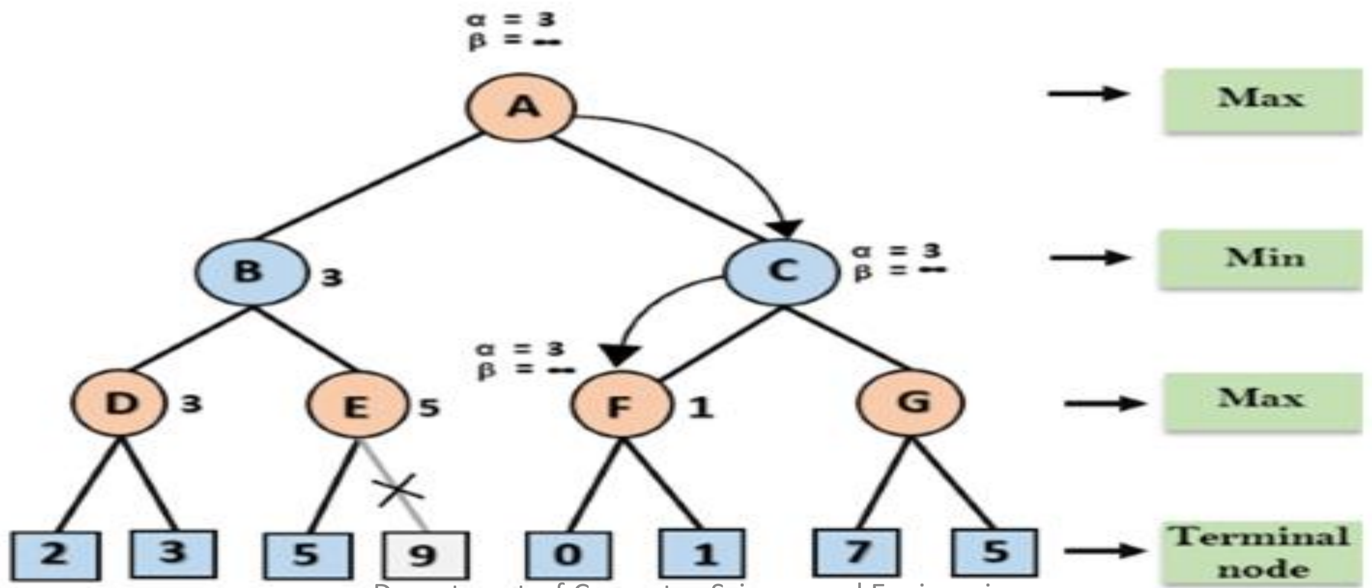
- **Step 2:** At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.
- **Step 3:** Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



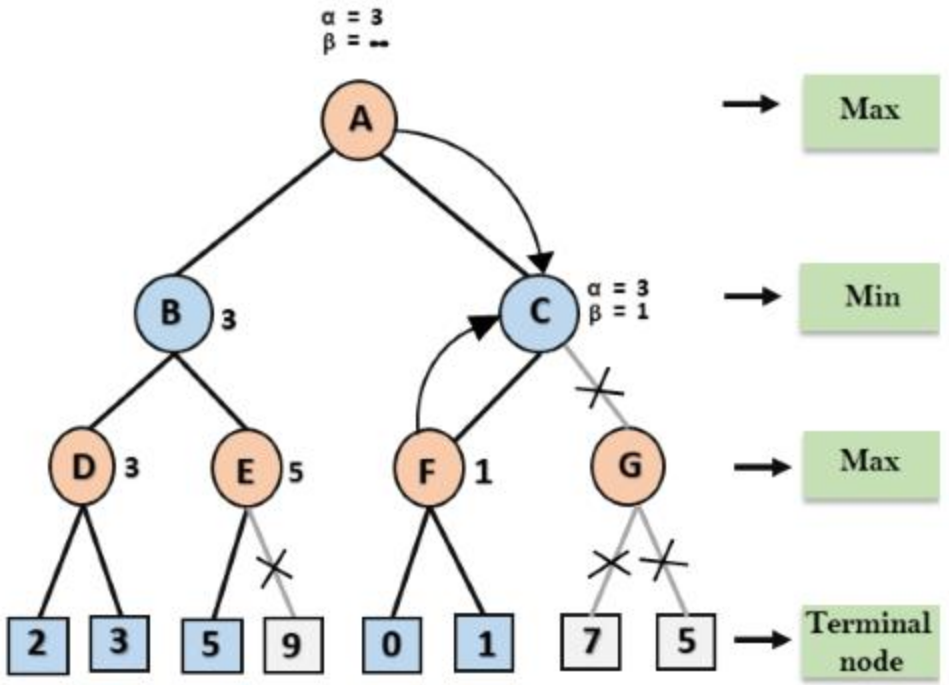
- **Step 4:** At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



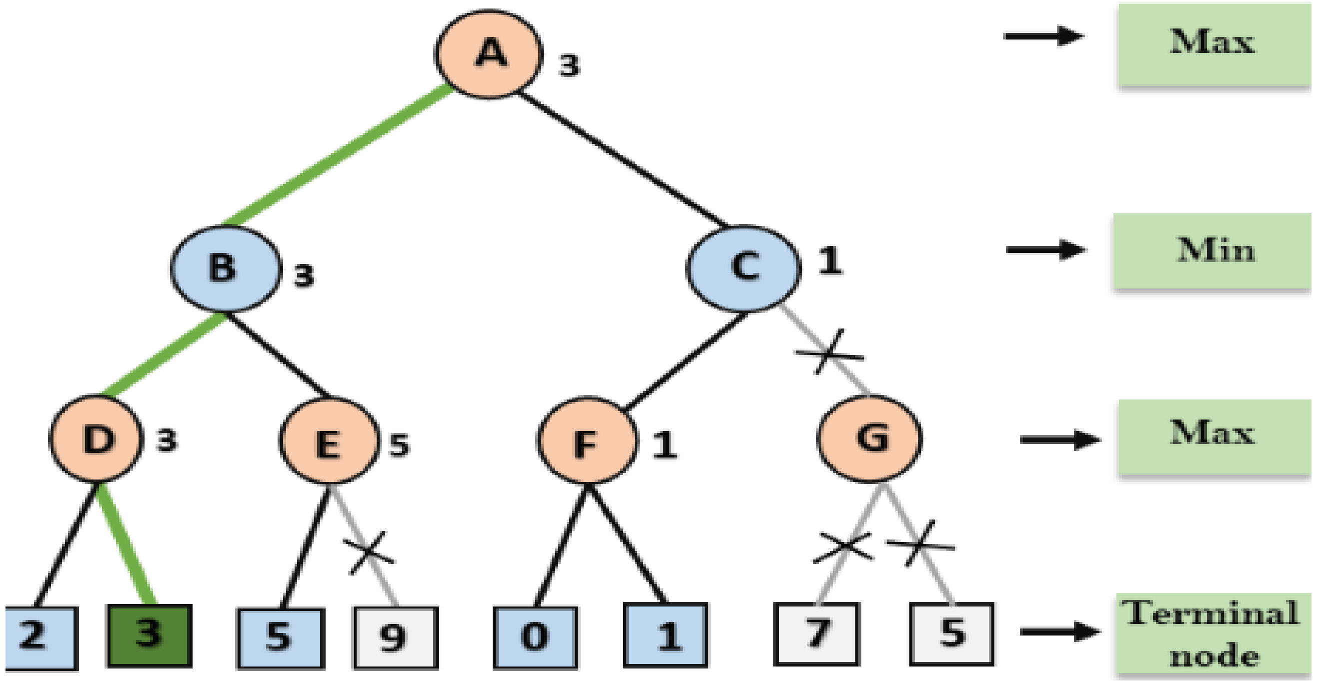
- **Step 5:** At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.
- At node C, α=3 and β= +∞, and the same values will be passed on to node F.
- **Step 6:** At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



- **Step 7:** Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



- **Step 8:** C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



```
function minimax(node, depth, alpha, beta, maximizingPlayer) is
if depth ==0 or node is a terminal node then
return static evaluation of node

if MaximizingPlayer then      // for Maximizer Player
maxEva= -infinity
for each child of node do
eva= minimax(child, depth-1, alpha, beta, False)
maxEva= max(maxEva, eva)
alpha= max(alpha, maxEva)
 if beta<=alpha
 break
return maxEva
else                          // for Minimizer player
minEva= +infinity
for each child of node do
eva= minimax(child, depth-1, alpha, beta, true)
 minEva= min(minEva, eva)
beta= min(beta, eva)
if beta<=alpha
break
return minEva
```

Game theory problems

- Prisoner’s Dilemma.

https://www.youtube.com/watch?v=t9Lo2fgxWHw

- Rock-Paper-Scissors
- Closed-bag exchange Game,
- The Friend or Foe Game, and
- The iterated Snowdrift Game.

RAMAPURAM
SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Thank you...