Unit 3 — 6 hrs.

Search Techniques

3.1 Uninformed Search Techniques:
Depth First Search (DFS), Breadth First Search (BFS), Depth Limit Search (DLS), and Search Strategy Comparison

3.2 Informed Search Techniques:
Hill Climbing, Best First Search (BFS), Greedy Search (GS), A* Search, Simulated Annealing, Genetic Algorithms

3.3 Adversarial Search Techniques:
Min-Max Procedure, Alpha-Beta Procedure

3.1 Uninformed Search Techniques:

Uninformed Search:

An uninformed search can also be called as **blind search**. This search technique has no information about its domain. The only thing that a blind search can do is distinguish a non-goal state from a goal state. *Depth First search* and *Bread First search* are good examples of Blind (or Uninformed) search strategies.

Uninformed search does not have additional information about states beyond problem definition. Uninformed search is a class of general-purpose search algorithms which operates in brute force-way.

Following are the different Uninformed Search Techniques:

a. Breadth-first search

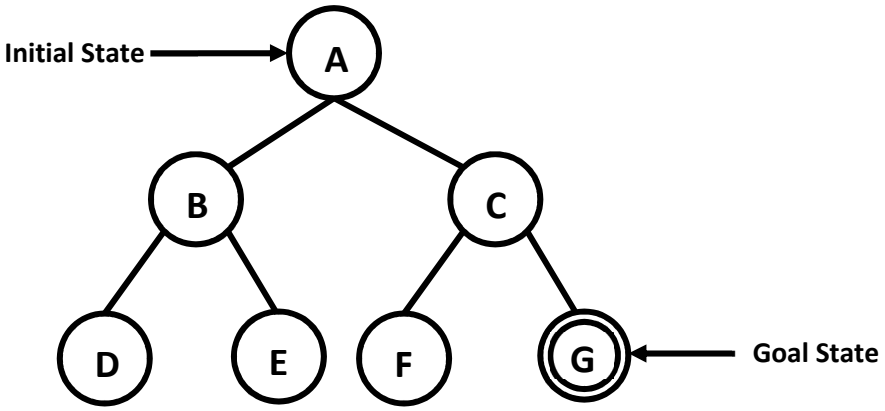
b. Depth-first search

c. Depth-limited search

d. Iterative deepening depth-first search

e. Bidirectional search

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]



- ⇨ We don't know the cost of start node to goal node.
- ⇨ We don't know the cost path of any node.
- ⇨ We don't know the steps to complete or reach to the goal state.
- ⇨ We just search blindly for reaching our goal state.

Depth First Search (DFS)

Depth-first search is a recursive algorithm for traversing a tree or graph data structure.

It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.

DFS uses a stack data structure for its implementation.

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

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.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Algorithm:

1. Start
2. Push root node to the stack
   - Mark root node as visited
   - Print root node in output sequence
3. Check Top stack
   - **If** (stack == Empty) **Goto Step 6**
4. Check adjacent of Top stack
   If (adjacent == not_visited)
   - Push node to the top of the stack
   - Mark node as visited
   - Print node in output sequence

   Else *i.e. (Adjacent == Visited)*
   - Pop- Top stack
5. Goto Step 3
6. Stop

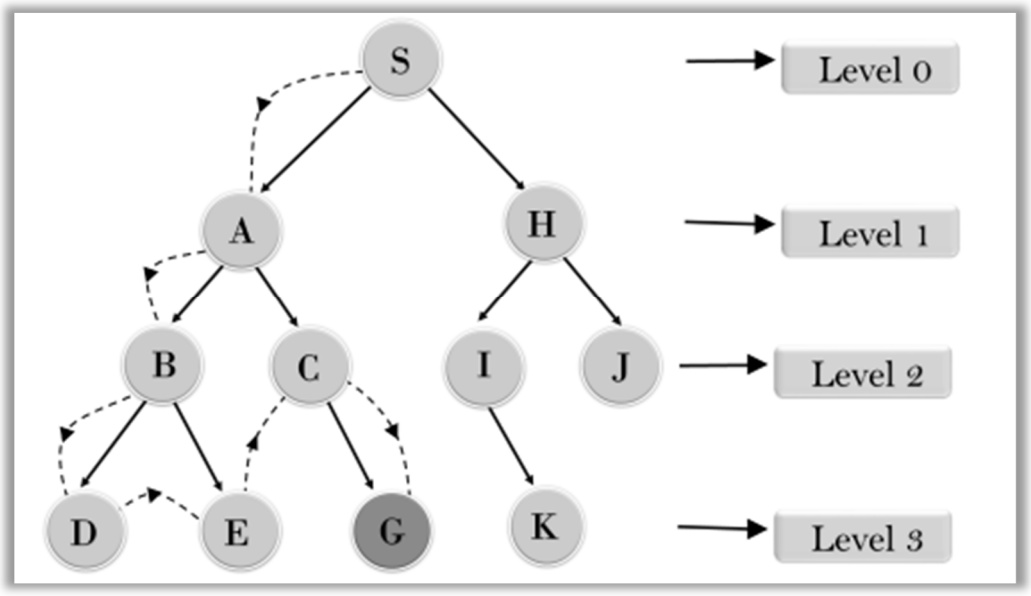
Example:

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----> D---> E----> C---> G



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.

Breadth First Search (BFS)

Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.

BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.

The breadth-first search algorithm is an example of a general-graph search algorithm.

Breadth-first search implemented using FIFO queue data structure.

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solution for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Algorithm:

1. Start
2. Point out the pointer to root node
   - Mark root node as visited
   - Print root node in output sequence
3. Search Pointer and Check for its adjacent nodes
   - **If** (Adjacent == Not_Visited)
     - Enqueue node into the Queue
     - Mark it as visited node
     - Goto Step 3

     Else *i.e. (Adjacent == Visited)*
     - Update the pointer to Top of the queue
     - Dequeue Top of the queue
     - Print node in output sequence
4. Check for the empty queue and pointer node has visited adjacent node

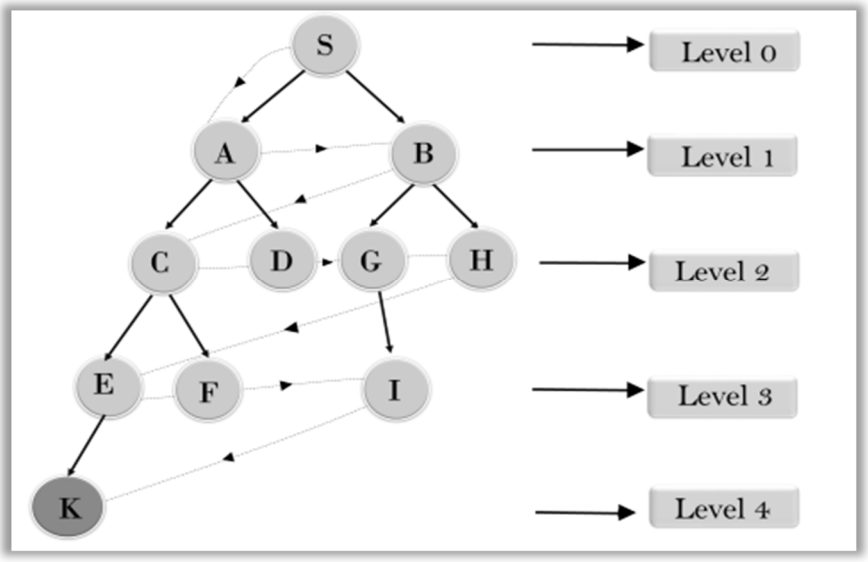
   If ((Queue == Empty) && (Pointer_Node_Adjacent == Visited))
   - Goto Step 5

   Else
   - Goto Step 3
5. Stop

Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S---> A---> B----> C---> D----> G---> H---> E----> F----> I----> K



© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

$T(b) = 1+b^2+b^3+.....+ b^d= O(b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

Depth Limit Search (DLS)

A depth-limited search algorithm 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.
- **Cutoff failure value:** It defines no solution for the problem within a given depth limit.

Advantages:

- Depth-limited search is Memory efficient.

Disadvantages:

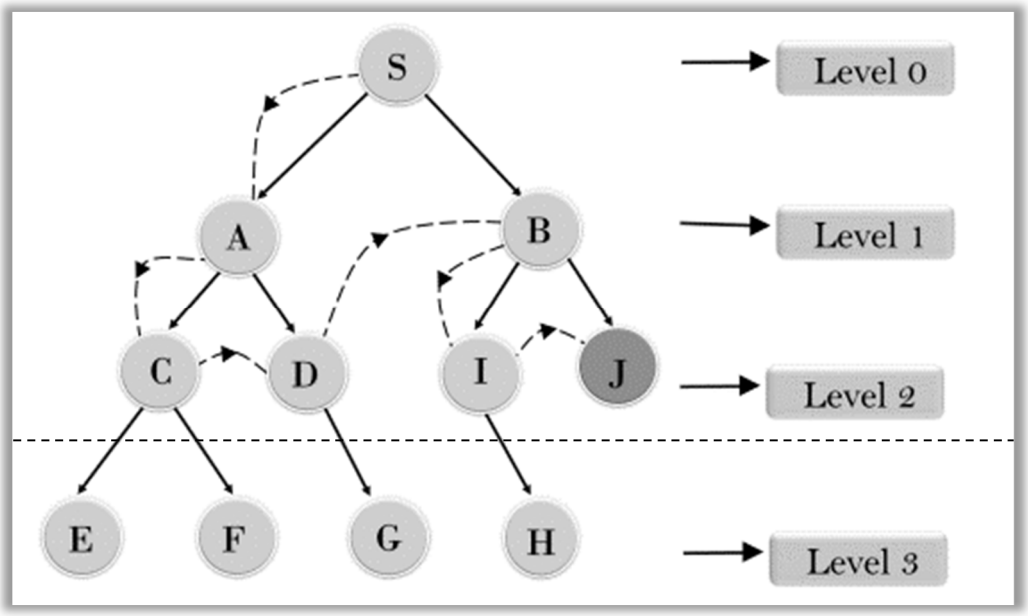
- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Example:

In the below tree structure, we have shown the traversing of the tree using DLS algorithm from the root node **S** to goal node **J**. DLS search algorithm traverse in depth and has some restriction i.e. limit here **limit=2** so, do not cross the limit beyond the dotted line, and the traversed path will be:

S---> A---> C----> D---> B----> I---> J

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is **O(b^{ℓ})**.

Space Complexity: Space complexity of DLS algorithm is O**($b \times \ell$)**.

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 (IDDFS)

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.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Advantages:

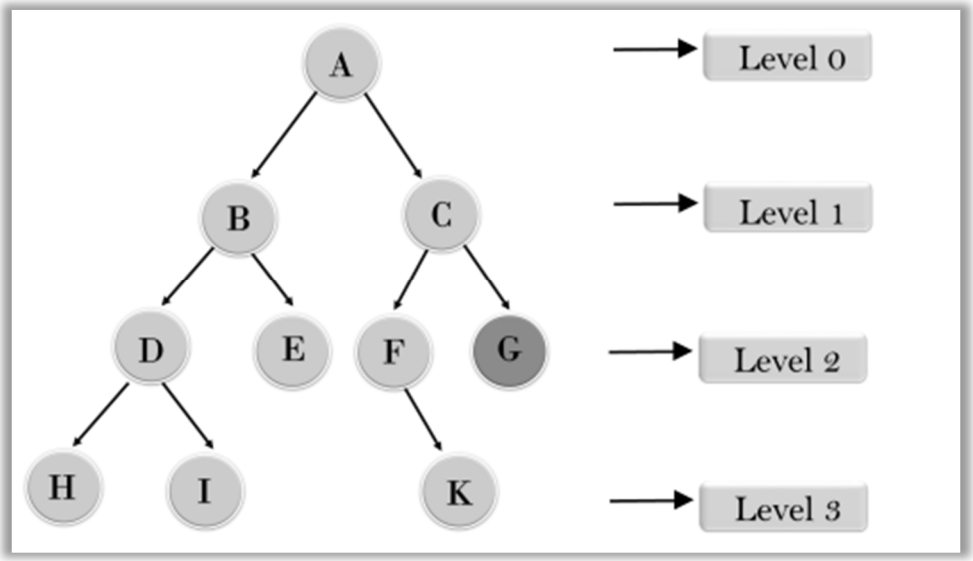
- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

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.

Completeness:

This algorithm is complete is if the branching factor is finite.

Time Complexity:

Let's suppose b is the branching factor and depth is d then the worst-case time complexity is $\mathbf{O(b^d)}$.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Space Complexity:

The space complexity of IDDFS will be **O(bd)**.

Optimal:

IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

Search Strategy Comparison

Criteria	BFS	DFS	Depth-Limit	IDS
Complete?	Yes	No	No	Yes
Time	$O(b^d)$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^d)$	O(bm)	O(bl)	O(bd)
Optimal?	Yes	No	No	Yes

Where,

b=Branching factor

d=Depth of the shallowest goal

m=Maximum depth of search tree

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

3.2 Informed Search Techniques:

Informed Search:

Informed Search is also known as Heuristic Search. Heuristic search is an AI search technique that employs heuristic for its moves. Heuristic is a rule of thumb that probably leads to a solution. Heuristics play a major role in search strategies because of exponential nature of the most problems. Heuristics helps to reduce the numbers of alternatives from an exponential number to a polynomial number.

In Artificial Intelligence, heuristic search a general meaning, and a more specialized technique meaning. In a general sense the term heuristic is used for any advice that is often effective, but is not guaranteed to work in every case. Within the heuristic search architecture, however, the term heuristic usually refers to the special case of a heuristic function.

Heuristic Algorithm:

The term heuristic is used for algorithms which find solutions among all possible ones, but they do not guarantee that the best will be found, therefore they may be considered as approximately and not accurate algorithms.

These algorithms, usually find a solution close to the best one and they find it fast and easily.

Some times these algorithms can be accurate, that is they actually find the best solution, but the algorithm is still called heuristic until this best solution is proven to be the best solution.

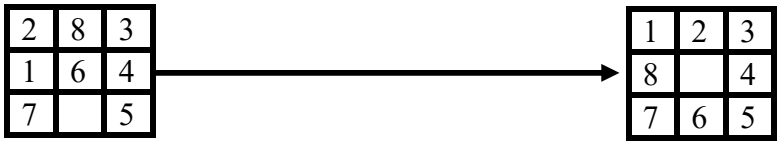
Summary of Informed / Heuristic Search:

- ⇨ Heuristic is derived from **heuriskein** in Greek, meaning ***"to find"*** or ***"to discover"***
- ⇨ Heuristic value will be associated to each node as an additional information to traverse the tree.
- ⇨ A solution cost estimation is used to guide the search.
- ⇨ The optimal solution, or even a solution is not guaranteed.
- ⇨ Based on h(n)- Estimated cost of path to goal ("Remaining path cost").
- ⇨ You cannot reach the node if you don't have additional information.
- ⇨ Heuristic Search which can give an optimized solution.
- ⇨ The purpose of heuristic function is to guide the search process in the most profitable path among all the available.
- ⇨ Artificial intelligence heuristic search AKA guided search. Heuristic in AI is a method that might not always find best solution but it guarantees to find a good solution in a reasonable time.
- ⇨ Heuristic in AI sacrifices completeness but increases efficiency.
- ⇨ Heuristic have heuristic function which estimates cost of path.
    - $\boldsymbol{f(n) = g(n) + h(n)}$

Where, $\boldsymbol{f(n) = Evaluation\ function\ or\ Total\ Path\ Cost}$

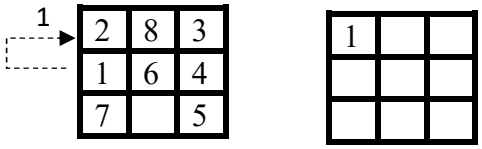
$\boldsymbol{g(n) = Local\ Path\ Cost}$ $\boldsymbol{h(n) = Heuristic\ Cost}$

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]



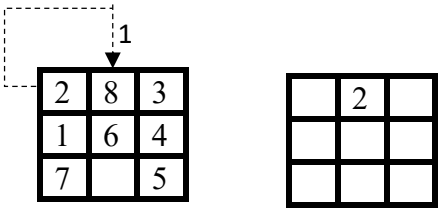
Start from number 1 :

Observe Initial State where is 1 located and try to match it with goal state, do a calculation at how many shift will I get the position of 1 like in the goal state *(for this you need to count the block either from the left, right or from the top, bottom)*. To shift 1 use always shortest path. At 1 shift it will reach to the goal state position. So, count it as 1 and write it in a sum of sequence.



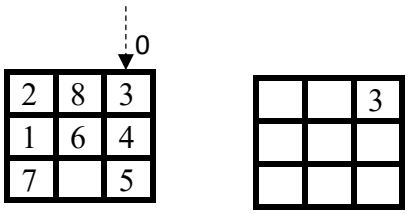
For number 2 :

Observe Initial State where is 2 located and try to match it with goal state, do a calculation at how many shift will I get the position of 2 like in the goal state *(for this you need to count the block either from the left, right or from the top, bottom)*. To shift 2 use always shortest path. At 1 shift it will reach to the goal state position. So, count it as 1 and write it in a sum of sequence.



For number 3 :

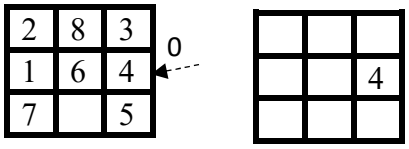
Observe Initial State where is 3 located and try to match it with goal state, do a calculation at how many shift will I get the position of 3 like in the goal state *(for this you need to count the block either from the left, right or from the top, bottom)*. To shift 3, use always shortest path. At 0 shift it will reach to the goal state position. No. 3 is already in the goal state. So, count it as 0 and write it in a sum of sequence.



For number 4 :

Observe Initial State where is 4 located and try to match it with goal state, do a calculation at how many shift will I get the position of 4 like in the goal state *(for this you need to count the block either from the left, right or from the top, bottom)*. To shift 4, use always shortest path. At 0 shift it will reach to the goal state position. No. 4 is already in the goal state. So, count it as 0 and write it in a sum of sequence.

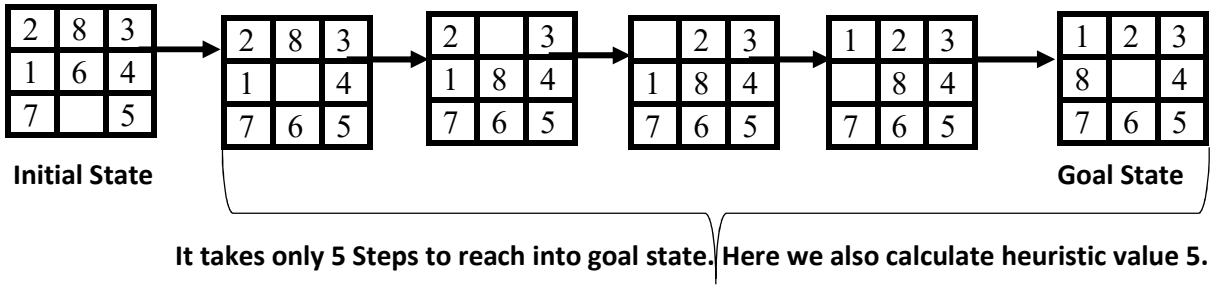
© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]



Like wise we do a calculation for remaining 5, 6, 7, 8 numbers

1 + 1 + 0 + 0 + 0 + 1 + 0 + 2 = 5

So, at 5 steps we'll reach at goal state. Here, h is the heuristic value.



Remember this will not always give optimal result, but here we got exact cost to reach into goal state.

Hill Climbing (HC)

- 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.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

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.

Types of Hill Climbing Algorithm:

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

1. 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

2. Steepest-Ascent hill climbing:

The steepest-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

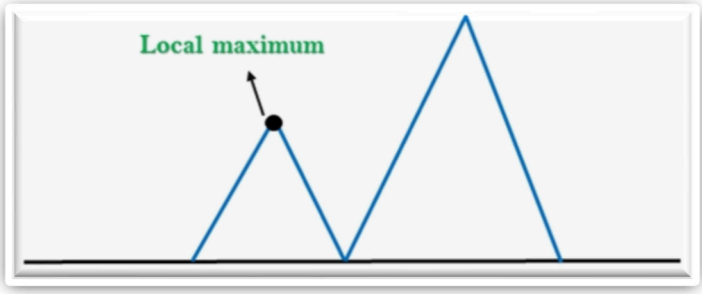
3. Stochastic hill climbing:

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.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

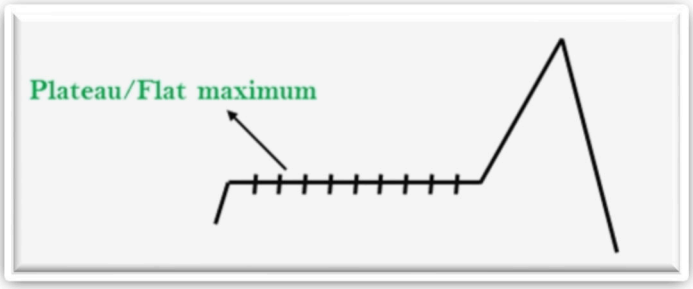
Limitations/Problems in Hill Climbing Algorithm:

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.



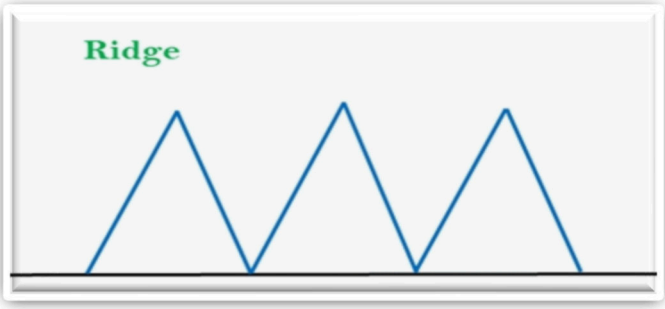
Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.

2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contain 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.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Algorithm:

1. Start
2. Evaluate the initial state
    a. If it is Goal state then Goto Step 6.
3. Loop until a solution is found or there are no new operators left.
4. Select and apply new operator.
5. Evaluate new state:
    a. If it is goal state, then Goto Step 6.
    b. Else If it is better than current state then
        i. Make it as New Current State.
    c. Else If it is not a better than current state then Goto Step 3.
6. Stop

Example:

Check out in previous chapter.

Hill Climbing example we already did in previous chapter **(8 Puzzle Problem).** To find a goal state at optimal (minimum) time.

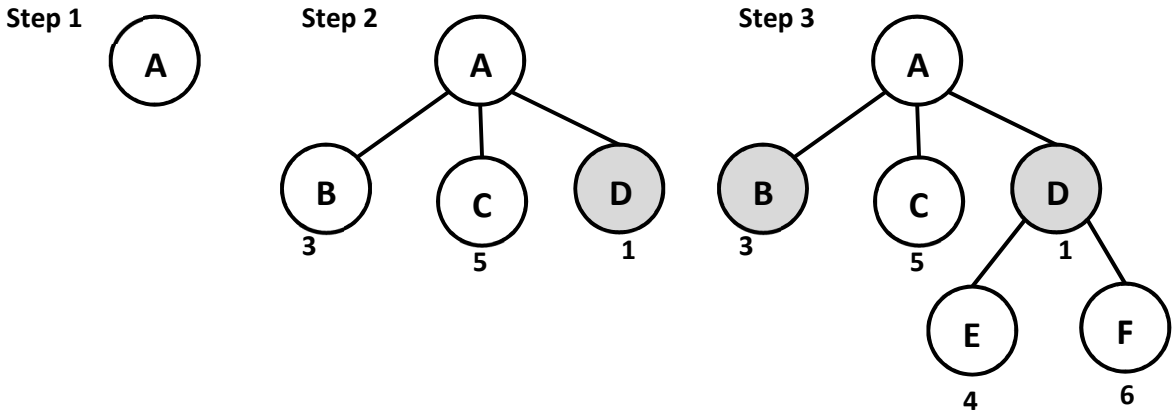
Best First Search (BFS)

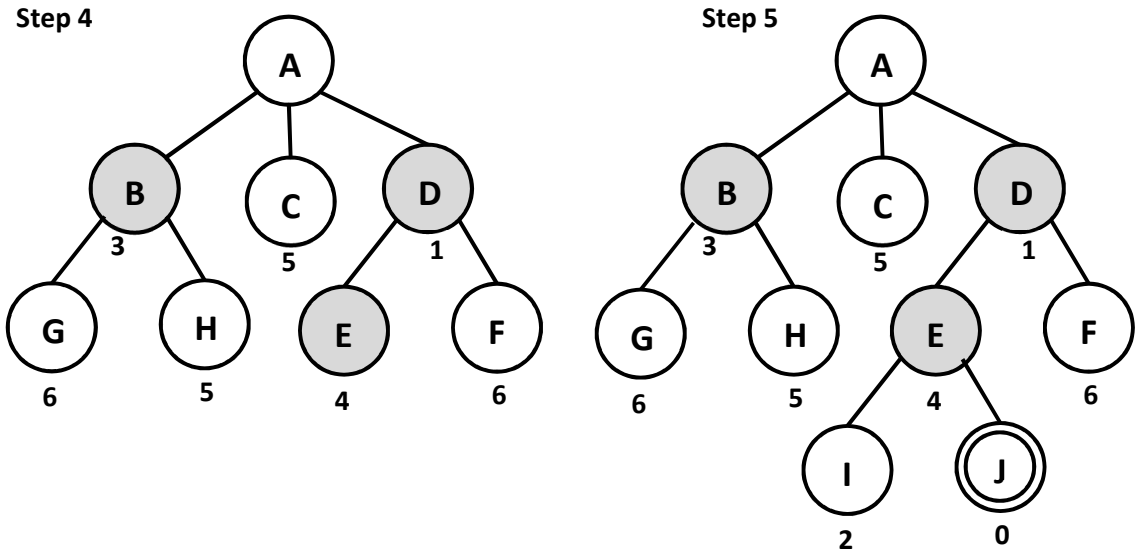
- ✓ A combination of **DFS** *(Higher Time Complexity)* and **BFS** *(Higher Space Complexity)*.
- ✓ At each step using an appropriate heuristic function, the most promising node is selected.
- ✓ If the node is a solution quit.
- ✓ Otherwise, add it to the set of nodes generated so far and carry on with generating and selecting the next node.
- ✓ Expand only those nodes which has least cost or less Evaluation function f(n).
- ✓ Best-First Search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.
- ✓ Best-First Search maintain Priority Queue.
- ✓ It has two list:
    - **Open List:** (Visited, Not Expanded/Explored)
        - Nodes that have been generated, but have not examined or expanded
    - **Close List:** (Visited, Expanded/Explored)
        - Nodes that have already been examined, whenever a new node is generated, check whether it has been generated before.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

- ✓ Every node in the search space has an evaluation function (heuristic function) associated with it.
- ✓ **Depth First Search:** Not all competing branches having to be expanded or **Backtrack.**
- ✓ **Breadth First Search:** Not getting trapped on dead-end paths or **No dead-end trap.**
- ✓ Combining the two is to follow a single path at a time, but switch paths whenever some competing path look more promising than the current one.

Simple Example:





© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Algorithm:

1. OPEN= {Initial State}
2. Loop until a goal is found or there are no node left in OPEN:
    a. Pick the best node in OPEN.
    b. Generate its Successors.
    c. For each Successor:
        i. New → Evaluate it, add it into OPEN, record its parent.
        ii. Generate Before → Change parent, Update Successors.

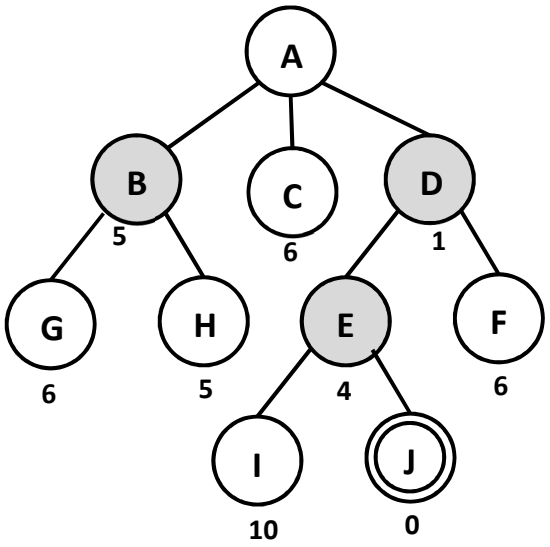
Advantages:

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst-case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

Example:



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 A and put in the CLOSED list

Initialization:

Open List: [D, B, C] **Closed List**: [A]

Iteration 1:

Open List: [E, B, C, F] **Closed List**: [A, D]

Iteration 2:

Open List: [J, B, C, F, I] **Closed List**: [A, D, E]

Iteration 3:

Open List: [B, C, F, I] **Closed List**: [A, D, E, **J**] *//Reach to the Goal State*

Hence the final solution path will be: **A----> D-----> E----> J**

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

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

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

Optimal: Best first search algorithm is not optimal.

Greedy Search (GS)

- A search method of selecting the best local choice at each step-in hope of finding an optimal solution.
- Does not consider how optimal the current solution is.
- At each step, uses a heuristic to estimate the distance (cost) of each local choice from the goal.
- A greedy algorithm always makes the choice that looks best at the moment.
- Greed makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.
- Greedy algorithms do not always yield/produce optimal solutions, but for some problems they do.
- You have to be greedy, so that best solution can be taken up.
- The decision is taken on the basis of current available information without worrying about the effect of current decision in future.
- **Feasible solution**, that may or may not **optimal solution**.
- **Feasible solution→** Any subset that satisfy given criteria (condition)
- **Optimal solution→** Best or most favorable solution (i.e. Maximum or Minimum Value).

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

- Select the best local choice at each step. NO matter what will happen in the upcoming nodes/future value.
- Once a decision has been made, it is never reconsidered.
- Obtain best solution than Best-First Search.
- But No guaranteed the optimum solution.

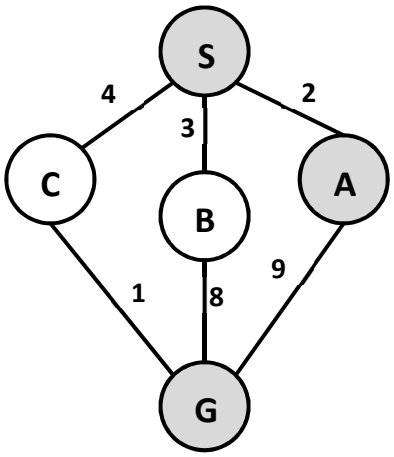
Features / Characteristics:

- ✓ To construct the solution in an optimal way. Algorithm maintains two sets:
  - One contains chosen items.
  - Another contains rejected items.
- ✓ Greedy algorithm makes good local choices
  - An optimal solution
  - Feasible solution

Areas of Applications:

- ✓ Finding Shortest Path.
- ✓ Finding Minimum Spanning Tree (Prim's or Kruskal Algorithms).
- ✓ Job Sequencing with deadline.
- ✓ Fractional Knapsack Problem.

Example:



S ----> A ----> G 2 + 9 = 11

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

A* Search

It is also known as A star. A* is the algorithm that is widely used in path finding and graph traversal, which is the process of finding a path between multiple points, called "nodes".

Worst case performance O(|E|) = O(b^d)
Worst case space complexity O(|V|) = O(b^d)

The A* algorithm combines features of uninformed cost search and pure heuristic search to efficiently compute optimal solutions. The A* algorithm is a Best First search algorithm in which the cost associated with a node is $\boldsymbol{f(n) = g(n) + h(n)}$

Where, $\boldsymbol{f(n) = Evaluation\ function\ or\ Total\ Path\ Cost}$

$\boldsymbol{g(n) = Local\ Path\ Cost}$ $\boldsymbol{h(n) = Heuristic\ Cost}$

A* is the best-known form of Best First Search. It avoids expanding paths that are already expensive, but expands most promising path first. A* algorithm guides an optimal path to a goal if the heuristic function $\boldsymbol{h(n)}$ is admissible, meaning it never overestimates actual costs.

Algorithm:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Advantages:

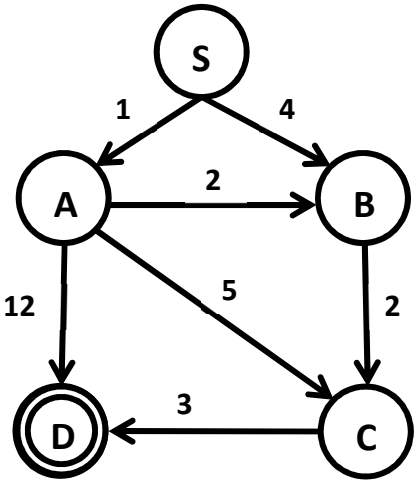
- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example:



Heuristic Values

S	7
A	6
B	2
C	1
D	0

Note:

D is a goal node so heuristic value is always 0 for every goal node.

Sol^{n}:

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

S →
= 0+7 = 7

S → A
= 1+6 = 7

S → B
= 4+2 = 6

S → A → B
= 1+2+2 = 5

S → A → C
= 1+5+1 = 7

S → A → D
= 1+12+0 = 13

S → A → B → C
= 1+2+2+1 = 6

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

S → A → B → C → D
= 1+2+2+3+0 = 8

S → B → C → D
= 4+2+3+0 = 9

S → A → C → D
= 1+5+3+0 = 9

Therefore, **S → A → B → C → D = 8**

We will follow the path to reach the goal state because it has low cost path, Hence, it is optimized solution.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So, the time complexity is **$O(b^d)$,** where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **$O(b^d)$**

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Simulated Annealing

A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum. And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient. **Simulated Annealing** is an algorithm which yields both efficiency and completeness.

In mechanical term **Annealing** is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state. The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move. If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.

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. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. 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.

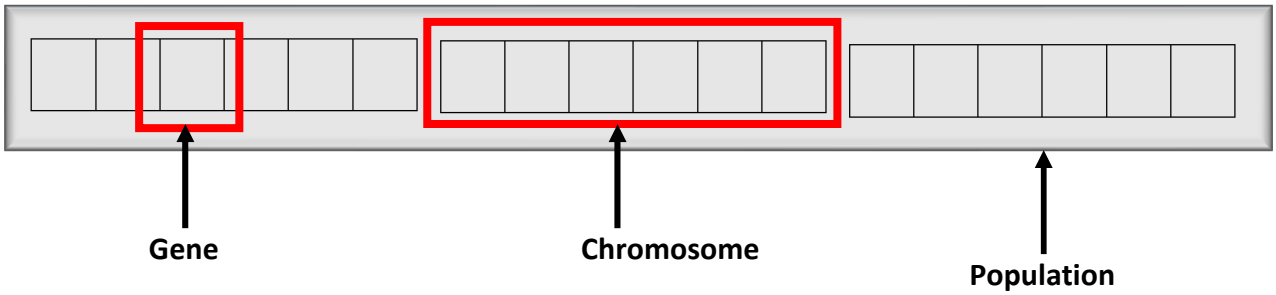
Foundation of Genetic Algorithms

Genetic algorithms are based on an analogy with genetic structure and behavior of chromosome of the population. Following is the foundation of GAs based on this analogy –

- Individual in population compete for resources and mate.
- Those individuals who are successful (fittest) then mate to create more offspring than others.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

- Genes from "fittest" parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
- Thus, each successive generation is more suited for their environment.



Fitness Score

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.

The GAs maintains the population of n individuals (chromosome/solutions) along with their fitness scores. The individuals having better fitness scores are given more chance to reproduce than others. The individuals with better fitness scores are selected who mate and produce **better offspring** by combining chromosomes of parents. The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating new generation when all the mating opportunity of the old population is exhausted. It is hoped that over successive generations better solutions will arrive while least fit die.

Each new generation has on average more **"better genes**" than the individual (solution) of previous generations. Thus, each new generation have better **"partial solutions"** than previous generations. Once the offspring produced having no significant difference than offspring produced by previous populations, the population is converged. The algorithm is said to be converted to a set of solutions for the problem.

Operators of Genetic Algorithms

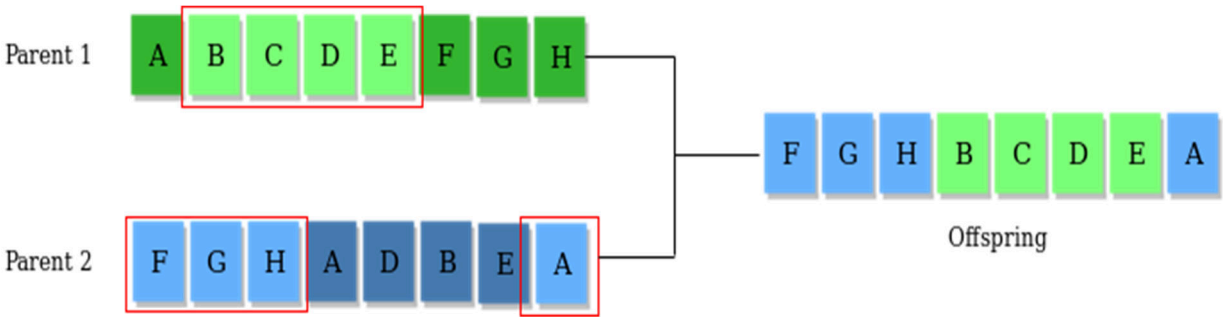
Once the initial generation is created, the algorithm evolves the generation using following operators –

1) Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to the successive generations.

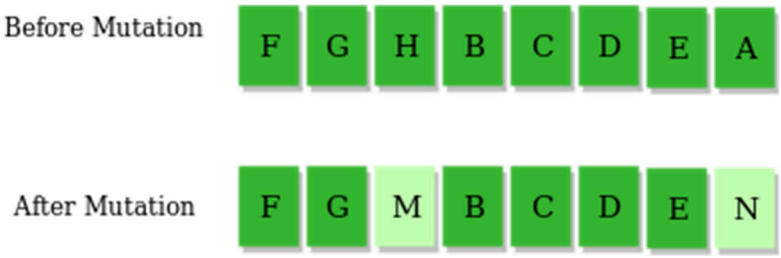
2) Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring).

For example –

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]



3) Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in population to avoid the premature convergence. For example –



The whole algorithm can be summarized as –

1) Randomly initialize populations p

2) Determine fitness of population

3) Until 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

Example problem and solution using Genetic Algorithms

Given a target string, the goal is to produce target string starting from a random string of the same length. In the following implementation, following analogies are made –

- Characters A-Z, a-z, 0-9 and other special symbols are considered as genes
- A string generated by these characters is considered as chromosome/solution/Individual

Fitness score is the number of characters which differ from characters in target string at a particular index. So individual having lower fitness value is given more preference.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Note: Every time algorithm starts with random strings, so output may differ

As we can see from the output, our algorithm sometimes stuck at a local optimum solution, this can be further improved by updating fitness score calculation algorithm or by tweaking mutation and crossover operators.

Why use Genetic Algorithms

- They are Robust
- Provide optimization over large space state.
- Unlike traditional AI, they do not break on slight change in input or presence of noise

Application of Genetic Algorithms

Genetic algorithms have many applications, some of them are –

- Recurrent Neural Network (RNN)
- Mutation testing
- Code breaking
- Filtering and signal processing
- Learning fuzzy rule base etc

3.3 Adversarial Search Techniques:

- In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the solution which often expressed in the form of a sequence of actions.
- But there might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.
- The environment with more than one agent is termed as multi-agent environment, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.
- So, 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.
- 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.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Just for Review *(We already discussed in Game Playing Topic)*

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

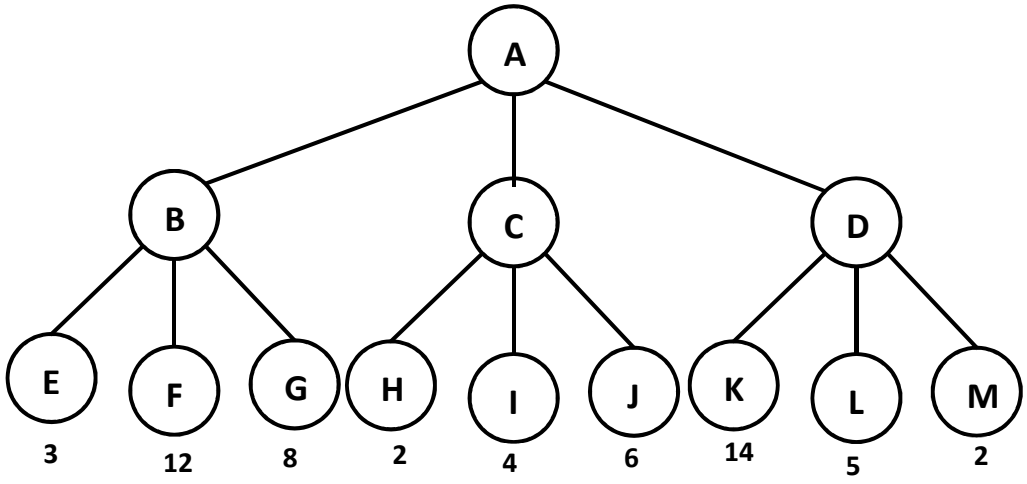
- **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.
- **Imperfect information:** If in a game agent do not have all information about the game and not aware with what's going on, such type of games is called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called as stochastic games. Example: Backgammon, Monopoly, Poker, etc.

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

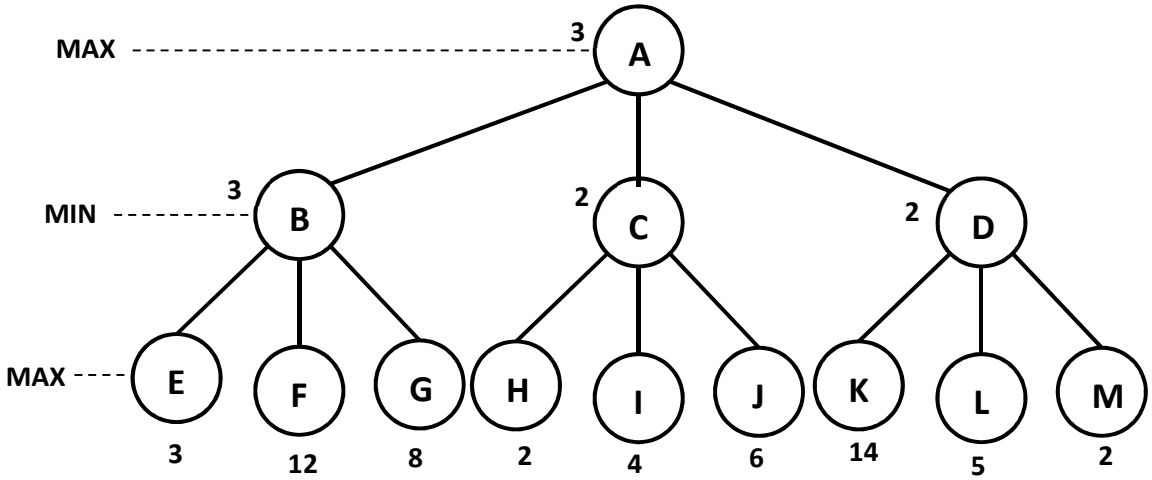
Min-Max Procedure

- Backtracking algorithm.
- Best move strategy used.
- MAX will try to maximize its utility (Best Move).
- MIN will try to minimize utility (Worst Move).
- Start with the current position as a MAX node.
- Expand the game tree a fixed number of ply (half-Moves).
- Apply the evolution function to the leaf position.
- Calculate back-up up values bottom-up.
- Pick the move which was chosen to give the MAX value at the root node.
- $\boldsymbol{MAX} = -\infty$ $\boldsymbol{MIN} = \infty$

Example:



Sol^{n}:



© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

$\boldsymbol{MIN = \infty}$

$(\infty, 3) = (3, 12) = (3, 8) = 3$

$(\infty, 2) = (2, 4) = (2, 6) = 2$

$(\infty, 14) = (14,5) = (5, 2) = 2$

$\boldsymbol{MAX = -\infty}$

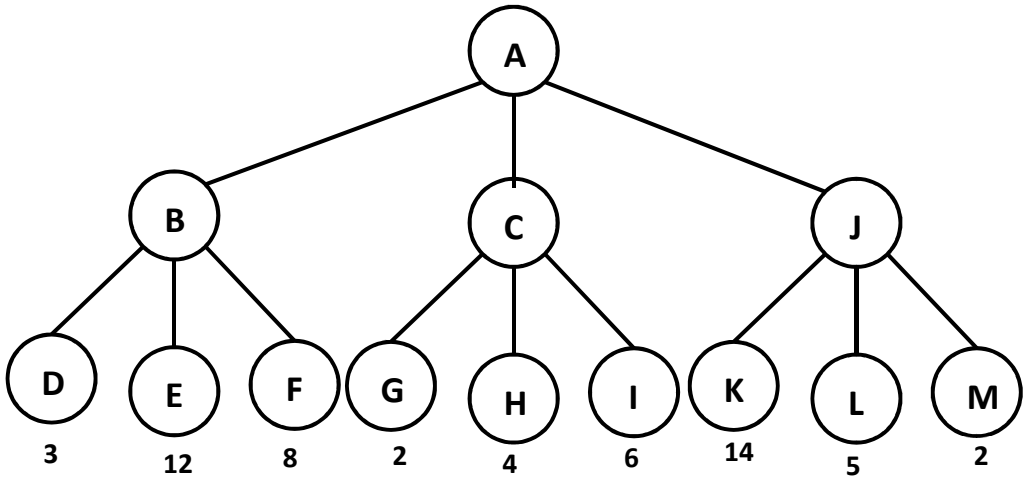
$(-\infty, 3) = (3, 2) = (3, 2) = 3$

Alpha-Beta Procedure

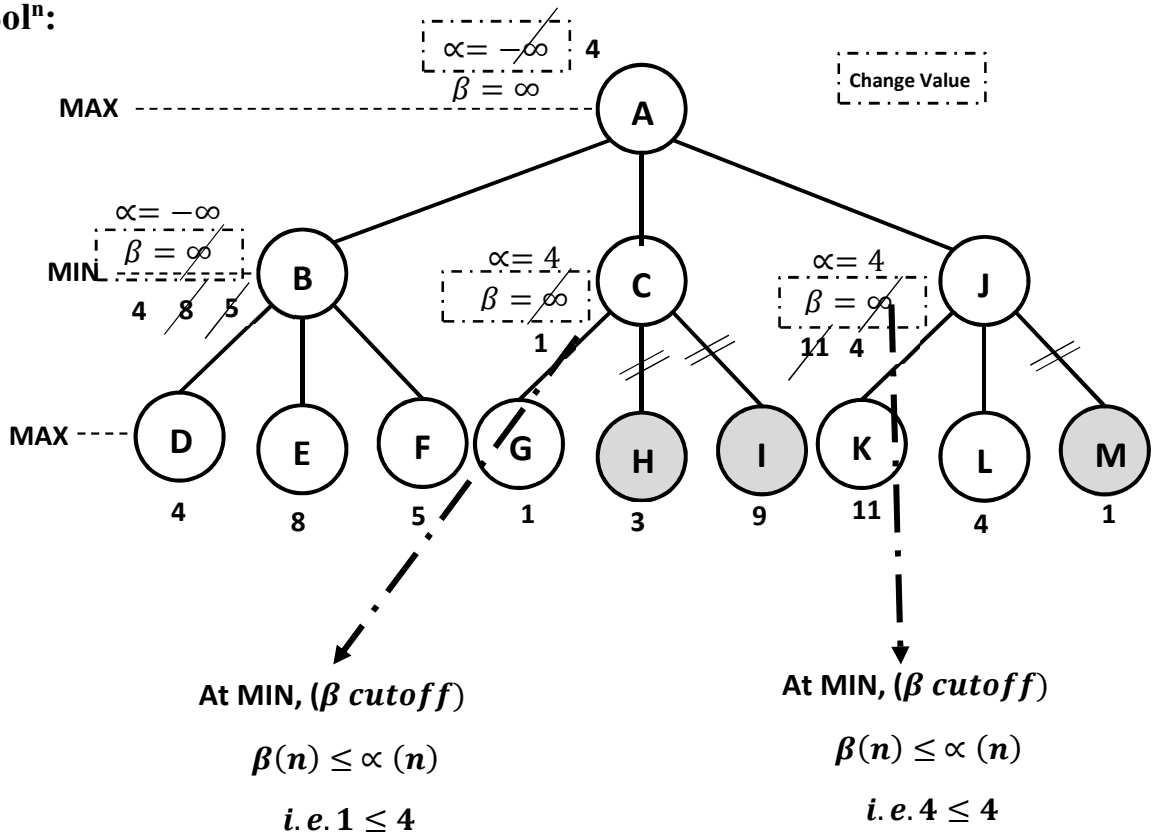
- A method that can often cutoff a half the game tree.
- Based on the idea that if a move is clearly bad, there is no need to follow the consequence of it.
- Alpha $(\propto) -$ Highest value we have found so far.
- Beta $(\beta) -$ Lowest value we have found so far.
- Traverse the search tree in Depth-First order.
- **MAX cut off rule:** At a MAX node n, cut off search if $\propto (\boldsymbol{n}) \geq \boldsymbol{\beta}(\boldsymbol{n})$
- **MIN cut off rule:** At a MIN node n, cut off search if $\boldsymbol{\beta}(\boldsymbol{n}) \leq \propto (\boldsymbol{n})$
- Pruning does not affect final results.
- Entire subtree can be pruning.
- **Pruning:** Way of degerming that certain branch which will do not be useful.
- **Alpha Cut** $(\propto) -$ If current MAX value is greater than the successor MIN value, don't explore that MIN sub tree any more.
- **Beta Cut** $(\beta) -$ If current MAX value is greater than the successor MIN value, don't explore that MIN sub tree any more.
- Alpha Beta Pruning is a method that reduces the number of nodes explored in Mini-Max strategy. It reduces the time required for the search.
- It is a depth-first search strategy. As root node is visited first then the sub tree.
- For the minimum nodes, the score computer start with **+ INFINITY** $+\infty$ and decreases with time.
- For the maximum nodes, the score computer start with **- INFINITY** $-\infty$ and increases with time.
-

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Example:



Sol^n:



© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]

Difference Between Uninformed and Informed Search

Un-Informed Search	Informed Search
1. Information is not available.	1. More Information is available.
2. Blind grouping is done.	2. Based on heuristic methods.
3. Search efficiency is low.	3. Searching is fast.
4. Impractical to solve very large problems.	4. Can handle large search problems
5. Best solution can be achieved.	5. Mostly a good enough solution is accepted as optimal solution.
6. Eg: DFS, BFS, IDS	6. Eg: Best First Search, A^*, Hill Climbing

Difference Between BFS and DFS Search

BFS	DFS
1. BFS stands for Breadth First Search	1. DFS stands for Depth First Search.
2. It uses the data structure queue.	2. It uses the data structure stack.
3. BFS traverse the tree level wise.	3. DFS traverse the tree depth wise.
4. BFS uses FIFO list.	4. DFS uses LIFO list.
5. BFS requires more memory.	5. DFS requires less memory
6. BFS is used to find shortest path.	6. DFS is used in cycle detection.
7. Always provide shallowest path solution.	7. Does not guarantee shallowest path Sol^n.
8. No backtracking is required in BFS.	8. Backtracking is required in DFS.
9. Never get trapped into finite loops.	9. Gets trapped into infinite loops.
10. Complete	10. Not Complete

THE END

© Deepak Bhatta Kaji (MCA)
Department of Computer Science – DEC [NAST]