Heuristic means an estimate i.e. anuman

UNIT-2 Heuristic Search Techniques

SYLLABUS

General search Algos :- Uninformed Search methods
- BFS, Uniform cost search

Depth First Search, Depth Limited Search, Iterative Deepening.

Informed search - Introduction, Generate and test, BFS, A* Search, Memory Bounded Heuristic search

Local search Algorithms and Optimization problems - Hill climbing and Simulated Annealing.

Uninformed Searching

1) Search without Information
2) No knowledge
3) Time Consuming
4) More Complexity (Time, Space)
5) DFS, BFS etc.

Informed Searching

1) Search with information
2) Use knowledge to find steps to solution
3) Quick solution
4) Less Complexity (Time, Space)
5) A*, Heuristic DFS, Best first Search

Uninformed search algorithms

Uninformed search means the machine blindly follows the algorithm regardless of whether right or wrong, efficient or in-efficient. These algorithms are brute force operations and they don't have extra information about the search space. The only information they have is on how to traverse or visit the nodes in the tree.

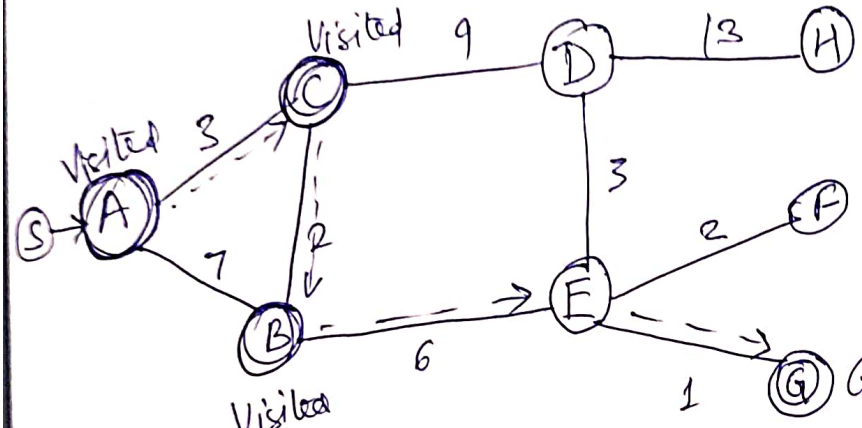
Thus, uninformed search algos. are also called blind search algos. The search algorithm produces the search tree without using any domain knowledge, which is a brute-force in nature. They don't have any background information on how to approach the goal or whatsoever.

Types :-

1. BFS (Breadth First Search)
2. DFS
3. Uniform Cost Search
4. Depth limited search.
5. Iterative Deepening DFS
6. Bidirectional search (if applicable)

Uniform Cost search Algorithm

1. Used for Weighted Tree / Graph Traversal
2. Goal is to find the path taking towards Goal node with lowest cumulative cost.
   (Try to find optimal path)
3. Node expansion is based on path costs.
4. Priority Queue is used for implementation.
   ↳ High Priority to min. cost
   ↳ Backtracking
5. Advantage: gives optimal solution (but not always) (because we choose min. cost path)
6. Disadv: Chances of infinite loop. It does not care about the no. of steps involved in searching & only concerned about path cost. Due to which it can stuck into infinite loop.



Path: $A \xrightarrow{3} C \xrightarrow{2} B \xrightarrow{6} E \xrightarrow{1} G$ = (12)
(without backtracking)

Now, if goal node is H,

Backtracking ↓ Back...

$A \xrightarrow{3} C \xrightarrow{2} B \xrightarrow{6} E \xrightarrow{1} G \xrightarrow{} E \xrightarrow{2} F \xrightarrow{2} E \xrightarrow{3} D \xrightarrow{13} H$

= (33)

We always see minimum cost path.

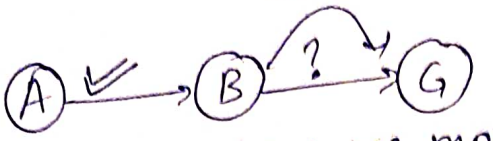
Time Complexity :- $O(b^{(C/\epsilon)})$ where ϵ is the lowest cost & C is the optimal cost

Space complexity :- $O(b^{(C/\varepsilon)})$

Optimal Yes (even for non-even cost)

Breadth First Search (BFS)

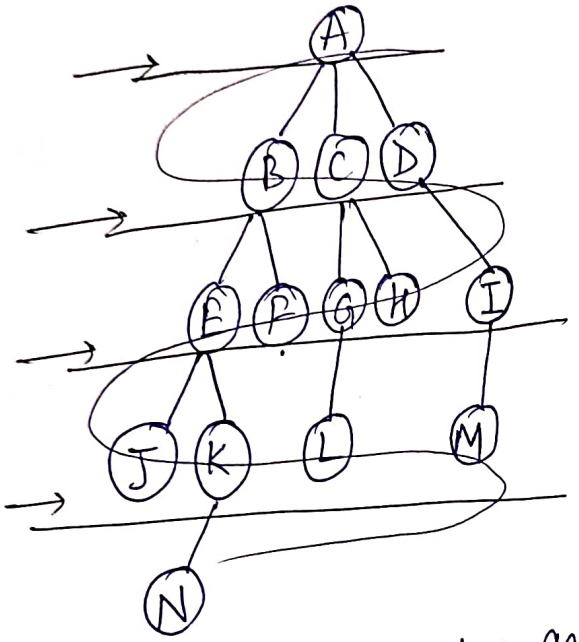
(Uninformed Search Technique) or
(Blind Search Technique) or
(Brute Force Technique)

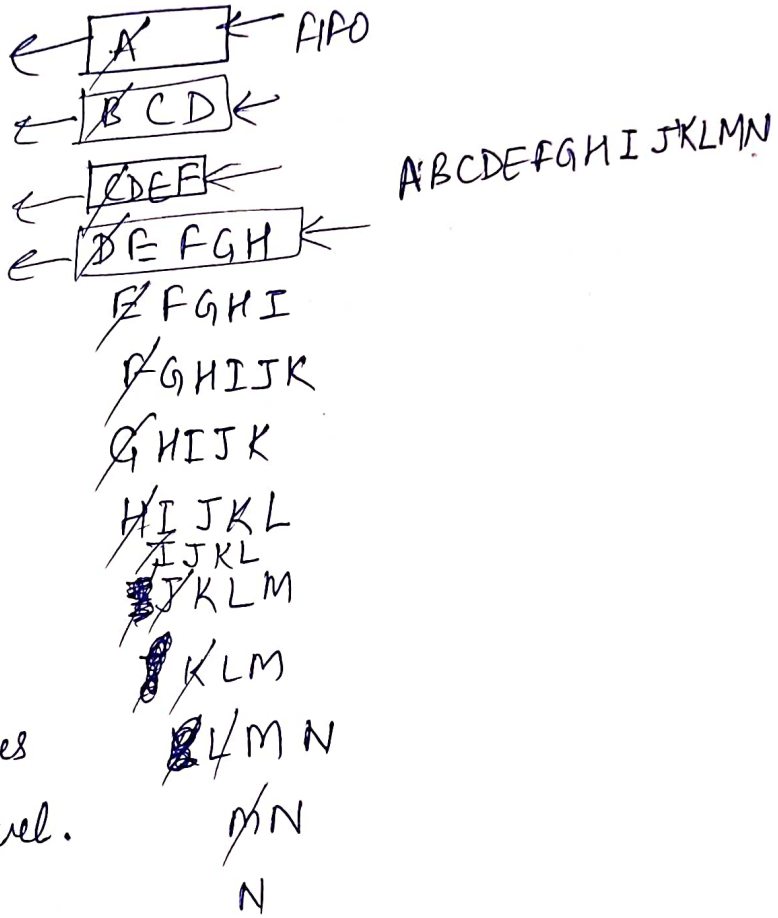


(cost)

We only know how we move from A state to B but don't know the cost from B to Goal state G. We don't know when we'll reach the goal state.

- Data structure used :- Queue
- Works on FIFO technique





It definitely searches an element because it traverses every element level by level.

It is COMPLETE.

It will give shortest path or least cost.

Time complexity = O(V+E) in D.S.

= $\boxed{O(b^d) \text{ in A.I.}}$ where b is the branch factor i.e. how many childrens

d is depth (level 0 to n)

(root node to childrens)

Breadth first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') & explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level.

Time Complexity : Equivalent to the no. of nodes traversed in BFS until the shallowest solution. $T(n) = 1 + n^2 + n^3 + \dots + n^s = O(n^s)$ (Lets say s)

Space Complexity : Equivalent to how large can the fringe get $S(n) = O(n^s)$

Completeness : BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.

Optimality : BFS is optimal as long as the costs of all edges are equal.

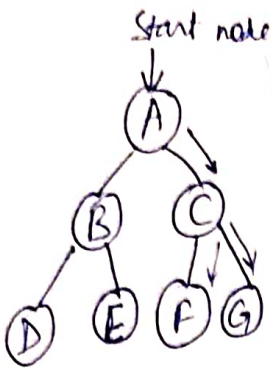
↳ Yes, if step cost = 1

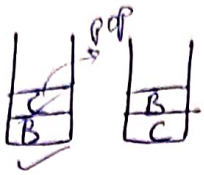
Shivani Kohli, MNIT

Depth First search

(Uninformed search Technique)

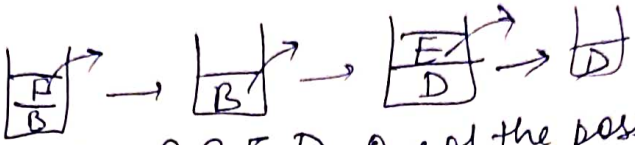
→ Deepest Node





Stack

LIFO

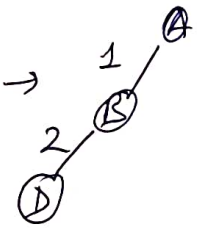


ACGFBED One of the possible DFS sequence

→ Non optimal

→ Incomplete (possibility not to give results if loop exists (∞) /or search space is very big)

$O(b^d)$ depth = 2



branching factor = how many children are possible of a node

DFS is a search algorithm where the search tree will be traversed from the root node. It will be traversing, searching for a key at the leaf of a particular branch. If the key is not found, the searching retraces its step back to the point from where the other branch was left unexplored & the same procedure is repeated for that other branch. The above image clearly explains the DFS algorithm. First, the searching starts in the root node A. Goes to the branch where node C is present. Then it goes to node G. But G has no any child nodes so we retrace the path in which we traversed earlier and again reach node C. Now there are two branches to traverse from which we have already traversed G.

Now we traverse F & then retrace the path as we have no further child nodes of F. Then we again traverse its parent node B and traverse its parent too. Finally, we reach the root node and repeat the same process for node A till last.

Advantages of DFS Algo:-

① It requires very little memory as it only needs to store a stack of the nodes on the path from the root node to the current node.
② It takes less time to reach the goal node (if traversed in right path)

Disadv. of DFS:-

① There is a possibility that many states keep reoccuring & there is not guarantee to reach the solution.
② It goes for Deep down searching & sometimes it may go to ∞ loop.
③ It takes lot of memory space & time to execute when the sol^{n} is at the bottom or end of the tree like BFS.

Time complexity $O(b^m)$

Space complexity $O(bm)$

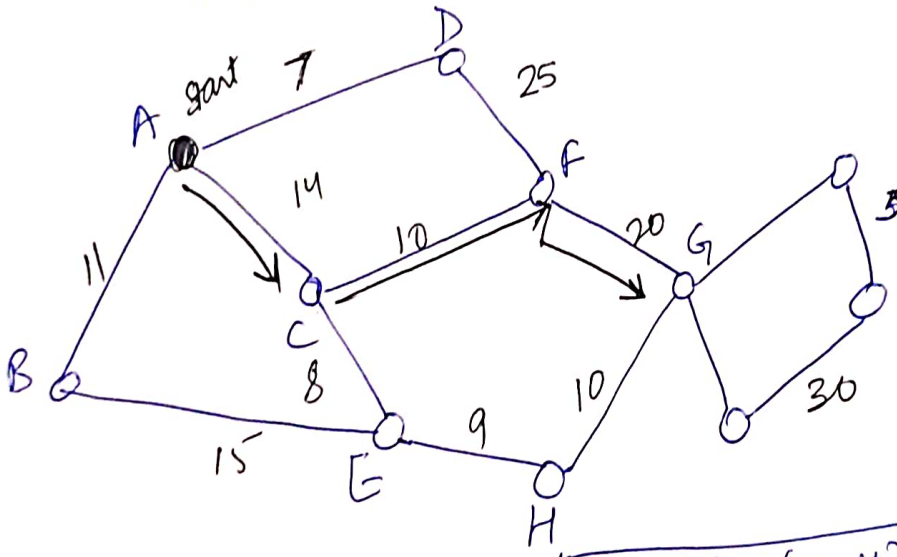
Optimality = Yes

Best First Search

Algo :- Let 'OPEN' be a priority queue containing initial state
↳ based on heuristic value of queue

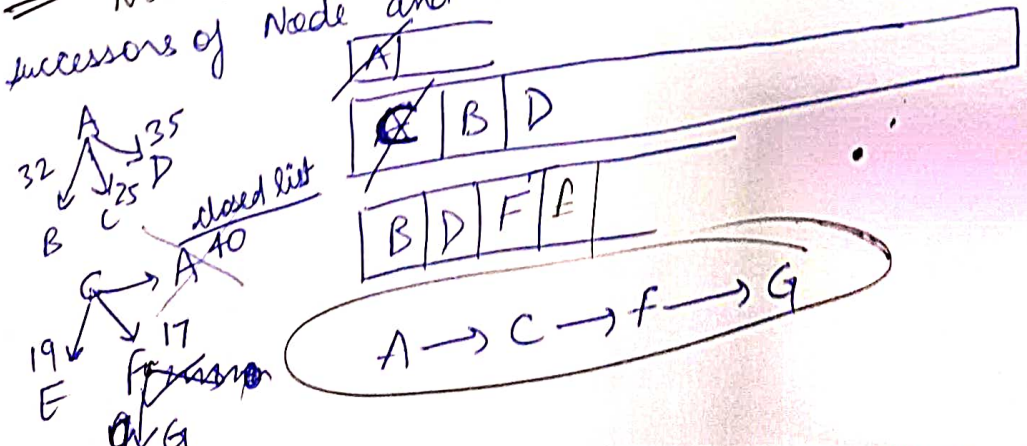
Loop
if OPEN is empty return failure
Node ← Remove-First (OPEN)
↳ if Node is a Goal
then return the path from initial to Node
else generate all successors of Node and
Put the newly generated Node into OPEN according to their f values
END LOOP

Straight line distances
A → G = 40
B → G = 32
C → G = 25
D → G = 35
E → G = 19
F → G = 17
H → G = 10
G → G = 0



Lets find out Euclidean distance $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$ (this is given here)

Sol^n OPEN = A
Now I will name A as Node & else part - generate all the successors of Node and Put the newly generated node into OPEN based on their f values



Now, lets consider if we calculate the path:-

$A \xrightarrow[14]{} C \xrightarrow[10]{} F \xrightarrow[20]{} G = 44$

$A \xrightarrow[14]{} \cancel{C} \xrightarrow[8]{} E \xrightarrow[9]{} H \xrightarrow[10]{} G = 41$

which is lesser than Heuristic value.

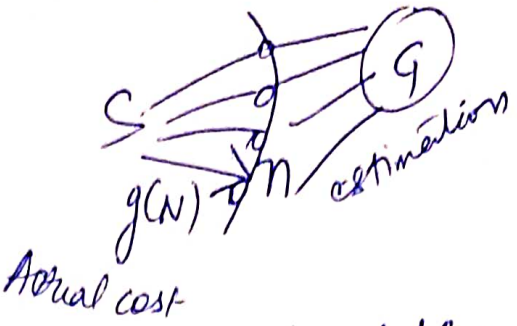
For this we introduce the use of A* algo here which provides the optimal solution taking into notice both possible path solutions.

A0*

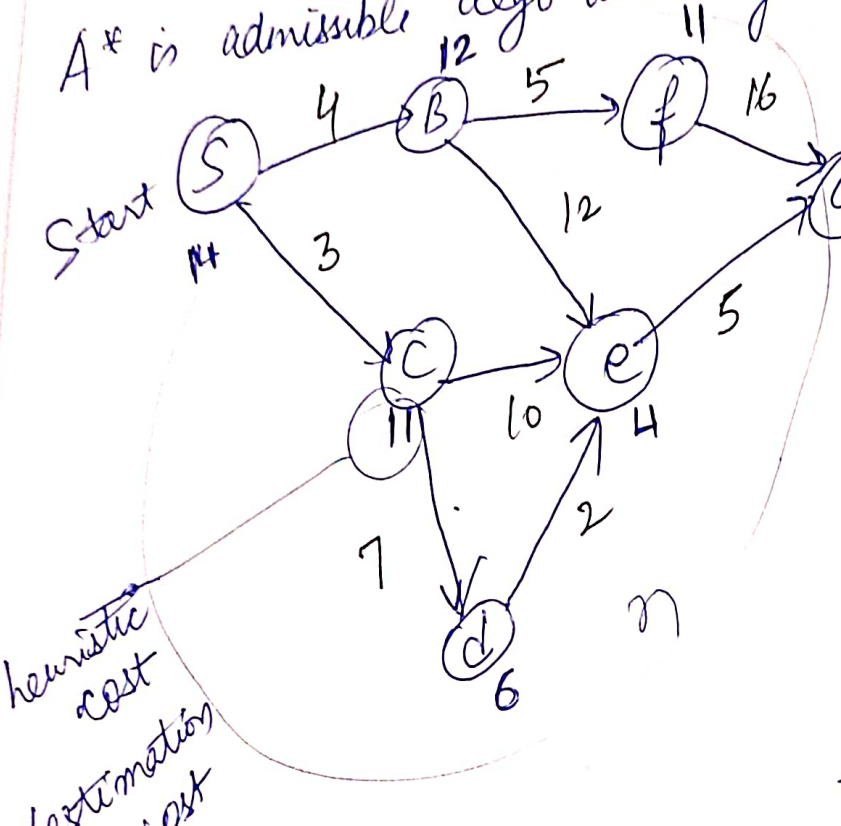
$f(N) = g(N) + h(N)$

↘ Estimation cost from n to goal node

↙ Actual cost from start node to n



A* is admissible algo which guarantees optimal solution.



Goal $S \to B$ or $S \to C$

$f(S) = 0 + 14 = 14$

$S \to B$ or $S \to C$

$4 + 12$ $\quad$ $3 + 11$

$= 16$ $\quad$ $= 14$

Select $S \to C$ ∵ $S \to C < S \to B$

$SC \to e$

$(3+10)+4$

$= 17$ ✓

$SC \to d$

$(3+7)+6$ ← heuristic value of d

$= 16$ ✓

$SCd \to e$ $\quad 3+7+2+4 = 16$

Compare $S \to B$ & $SC \to d$ (∵ both are 16)

$SB \to f$ $\quad 4+5+11 = 20$

$SB \to e$ $\quad 4+12+4 = 20$] ×

$SCde \to G$

$3+7+2+5+0 = 17$

SCde G

to reach goal state

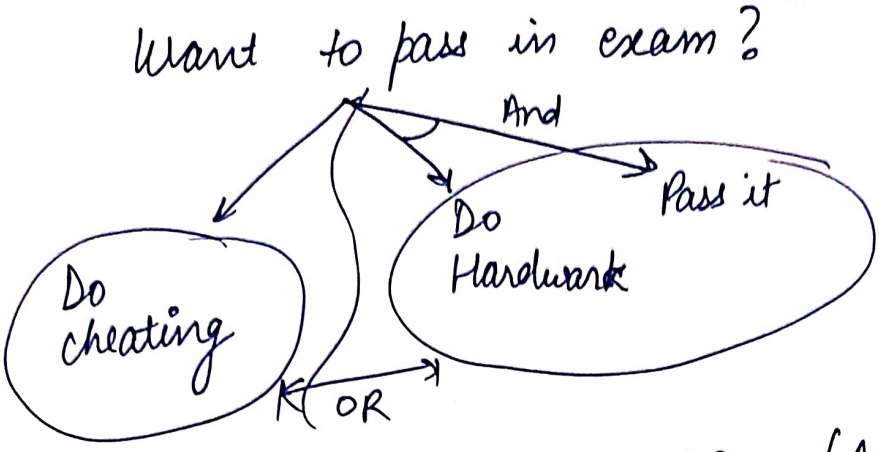
You can check with SBf → G

9 + 16 + 0 = (25)

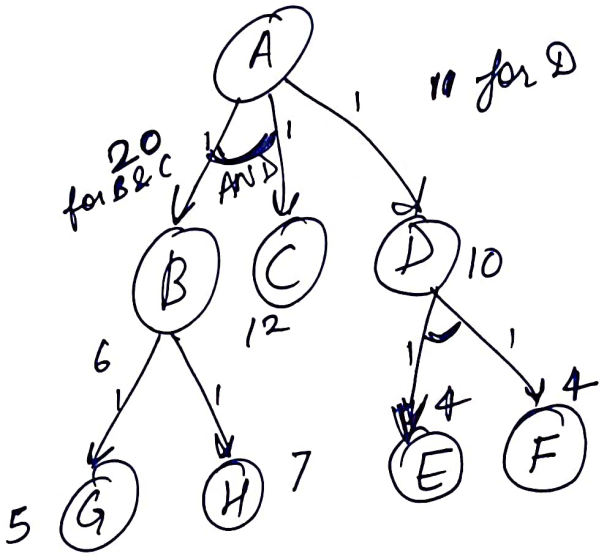
We will not take this

Final ans → SCdEG = (17)

AO* (AND/OR) → Problem Decomposition (Breakdown into smaller pieces)



A* always gives optimal solution. (Admissible)
AO* does not guarantee optimal soln.
It does not explore all the soln paths once it got a soln



1+1+6+12 = 20

1+10 = 11

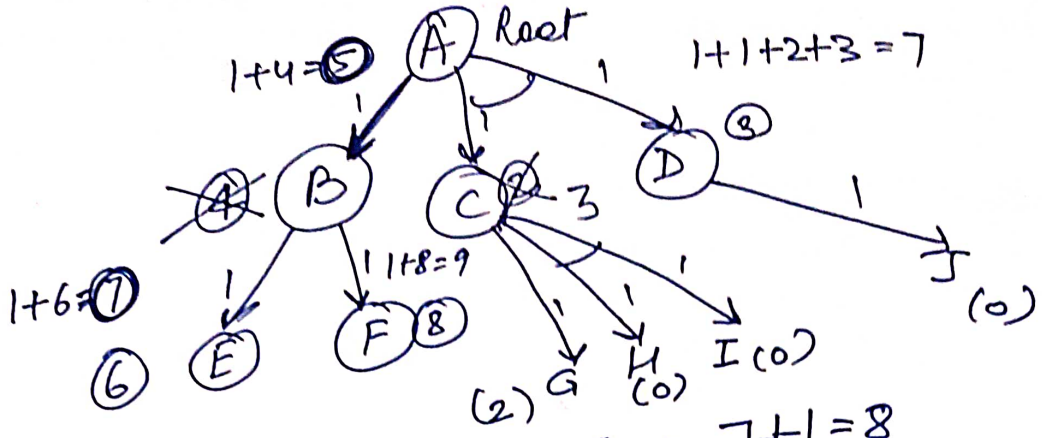
Choose 11

D to E & F = 1+1+4+4
= 10

A to leaf node = 10

A didnot explore after B & C

If it explored may be it reached optimal soln.



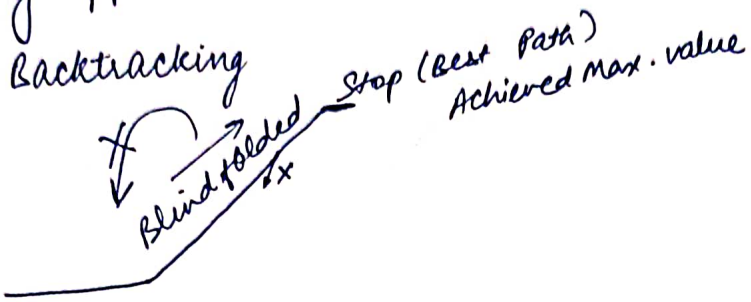
New Heuristic value of A = 7+1 = 8

To overcome the Iterative Deepening A* algo, memory req. Simplified memory Bound A* stores all the available memory with lowest cost.

→ Utilizes all the avail. memory efficiently.
→ Avoids repeated states (
→ It is complete if avail. memory is sufficient enough to store the shallowest path solution.
→ It is optimal
→ When enough memory is available for search, then it is optimally efficient.

Simple Hill Climbing Algorithm

- Local Search Algo
- Greedy Approach
- No Backtracking
- Only one best move



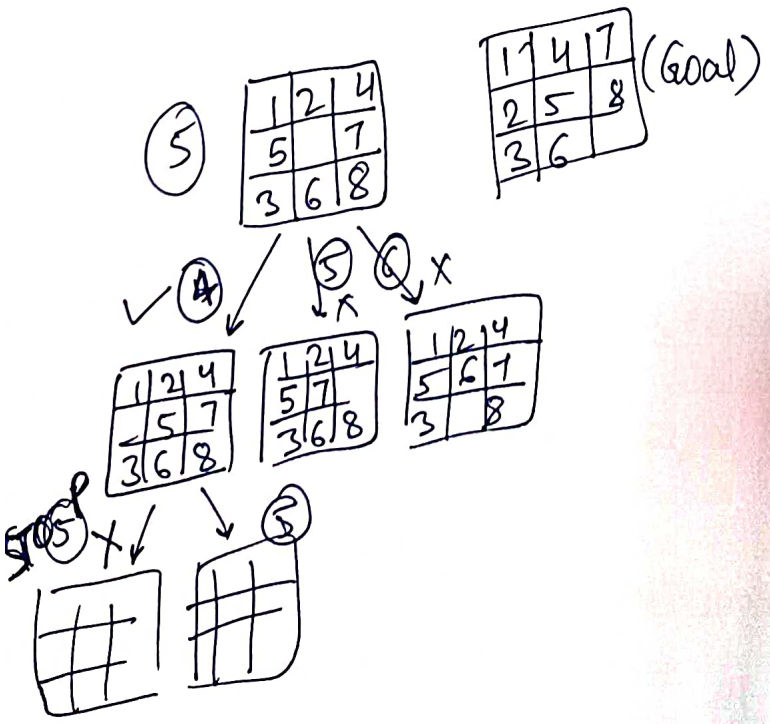
Step1 Evaluate the initial state

Step2 loop until a solution is found or there are no operators left

- Select and Apply a new operator
- Evaluate the new state :

if goal state achieved ; QUIT

if better than current state then it is new current state



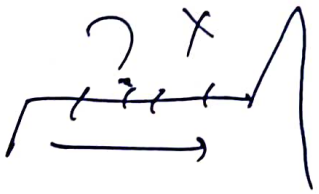
Problems

① Local Maximum

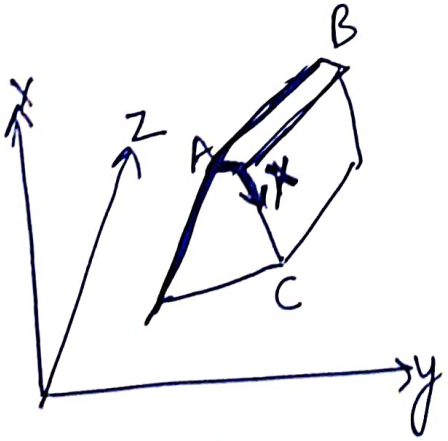
It will not go deep again bcoz it has achieved best value (Maxima)
But globally it is not Maximum.
It is only local Maximum.

② Plateu / Flat Maximum

(All neighbours have same value)



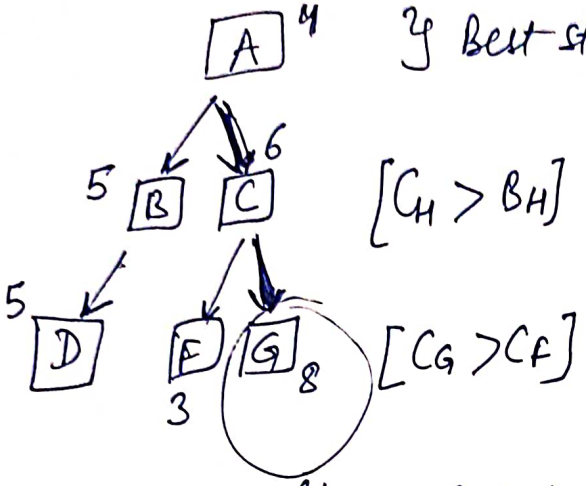
③ Ridge



Doesn't move in other direction to B. Have to go to C only but will not go to C bcoz it has achieved max. value.

Steepest Ascent Hill Climbing

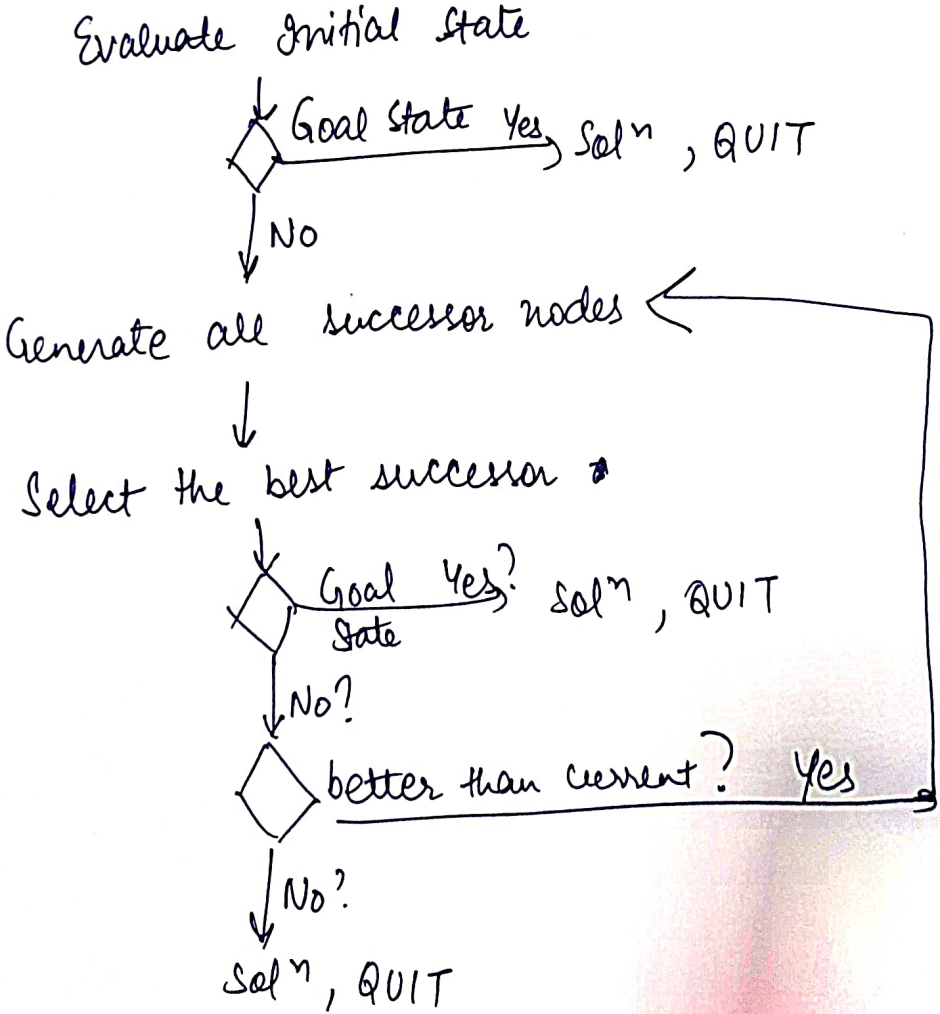
- Multiple Points are checked in Steepest Ascent Hill Climbing.
- All moves are considered & best one is selected as next state
- Examines all neighbouring nodes & selects nodes closest to solution as next node



If Best state is node with better heuristic value (>=)

Steepest Ascent → But slower than Simple Hill Climb

But better result

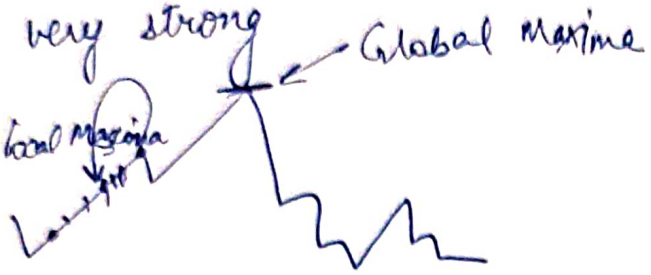


Simulated Annealing

Local Search algo.

Allows downward steps

↳ Annealing is a process in metallurgy where metals are slowly cooled to make them reach a state of low energy where they are very strong



- Checks all the neighbours
- Easy to code for complex problems also
- Gives good solution
- ~~Statistically~~ guarantees to find an optimal sol^n.

But

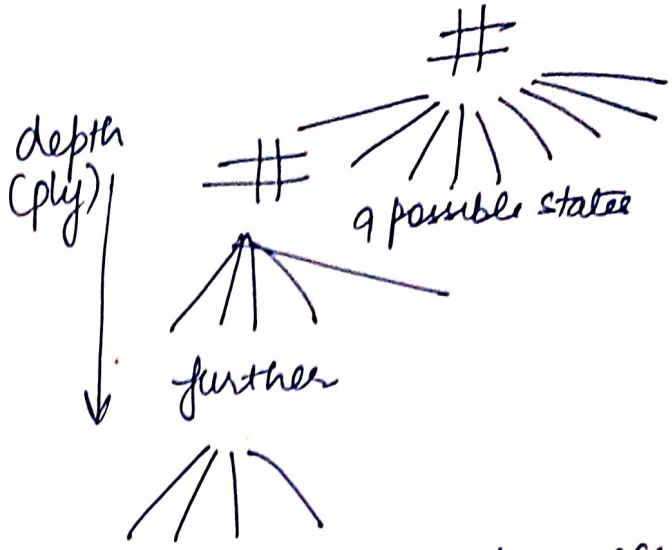
- Slow process
- Can't tell whether an optimal sol^n is found.
- Some other method is also required in this case

SA	VS	HC
• Annealing schedule is maintained		NIL
• Moves to worst states may be accepted (Small Backtrack)		NO
• Best state found so far is also maintained		YES

Game Playing

INTRODUCTION :-

1. Minimax algo
2. α-β pruning



Utility / pay off (credits)

game Tree

Search tree

Max (win prob.)
Min

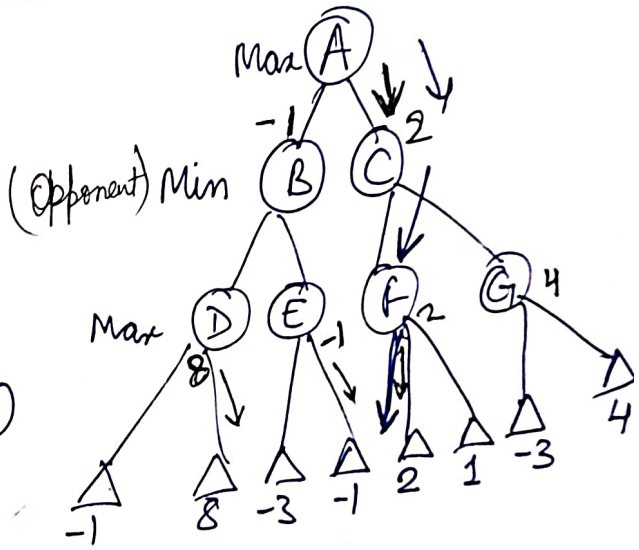
space graph

∴ We don't use BFS in game playing

MINIMAX ALGO

Shivani Rohilla
MNIT, Jaipur

1. Backtracking algorithm
2. Best move strategy
3. Max will try to maximize its utility (Best Move)
4. Min will try to minimize its utility (Worst Move for me)



Min will choose B to E because it will try to minimize the utility of Max.

Min will try to choose C to F because it has less utility

Max will choose A to C because it has more utility (2)

A → C → F → 2

GOAL :- TO FIND BEST PATH FOR MAX.

Time complexity of Minimax Algo $= O(b^d)$

$= 3^2 = 9$ (feasible)

But in case of chess, we have 35 choices possible

No. of moves = 50 (on a avg.)
(Depth) Ply

game tree $= 35^{(100)}$

We apply minimax algo in simple & small games only.

To bring efficiency into Minimax Algo, we follow Alpha-Beta Pruning. Like we don't need to explore all nodes, pruning the non-desired ones.

Alpha (generally for Max)

Beta (" " Min)

$O(b^{d/2})$ In best case or average case.

$O(b^d)$ in worst case

No. of nodes are explored only