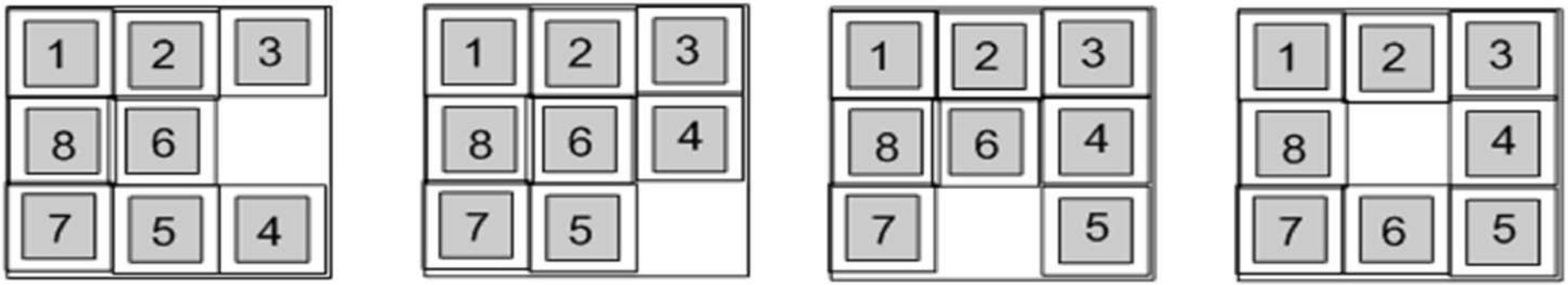
# Artificial Intelligence

## Lecture 28

# Problem Solving by Search

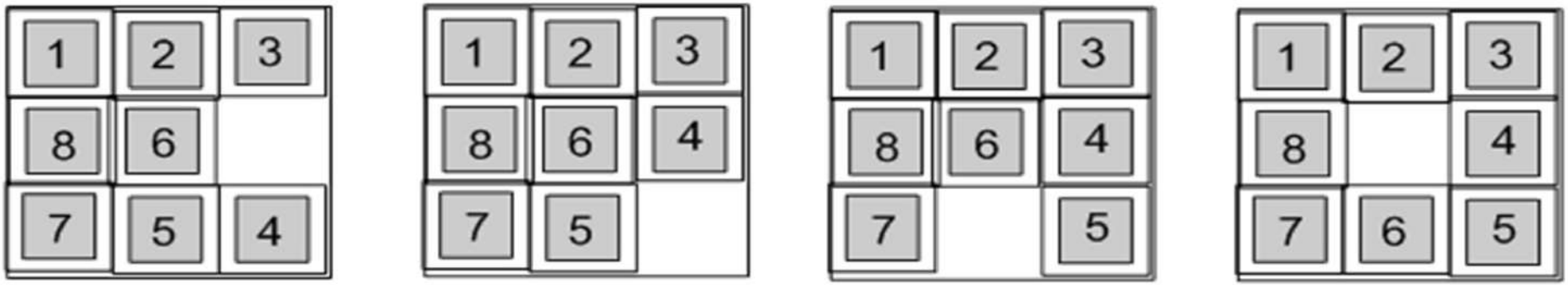
*Prepared by:*

**Md. Mijanur Rahman, Prof. Dr.**
Dept. of CSE, JKKNIU
Email: mijanjkkniu@gmail.com



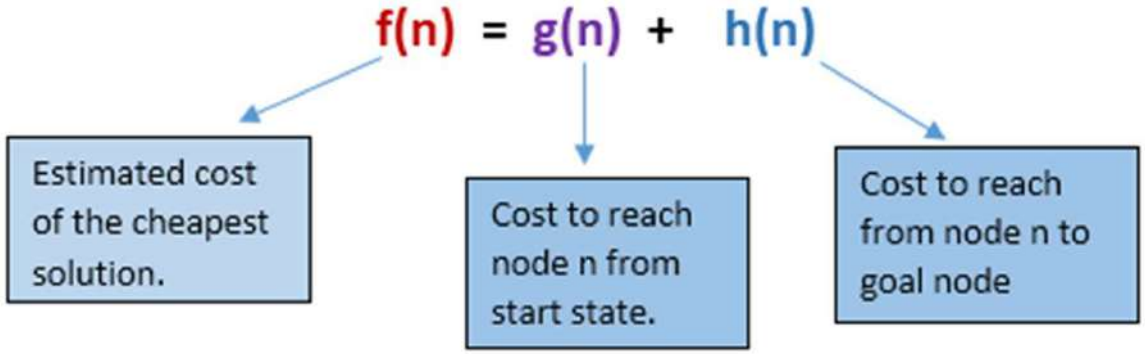
# Lecture Outlines

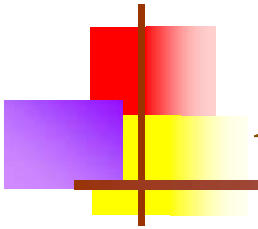
- **Tree searches and search strategies:**
  - Informed or Heuristic Search
    - Best First Search Algorithm (Greedy search)
    - **A* Search Algorithm**
    - **Hill Climbing Algorithm**



# A* Search Algorithm…

- A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost g(n) to reach the node n from the start state.
- It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function.
- This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).
- In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.





# A* Search Algorithm…

- **Algorithm of A* Search**

**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:

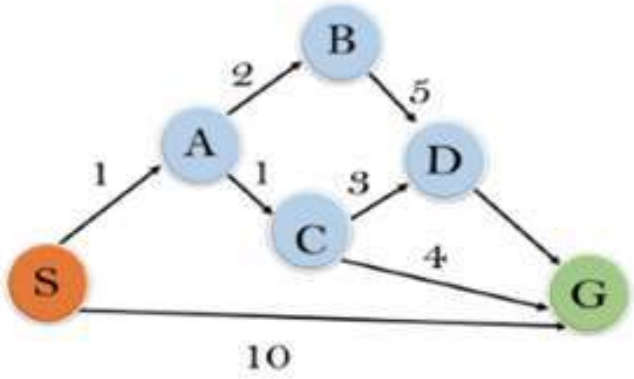
**Step 4:** Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

**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**.

# A* Search Algorithm…

- Example:
- In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.
- Here we will use OPEN and CLOSED list.




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

# A* Search Algorithm…

- Solution:

**Initialization:** {(S, 5)}

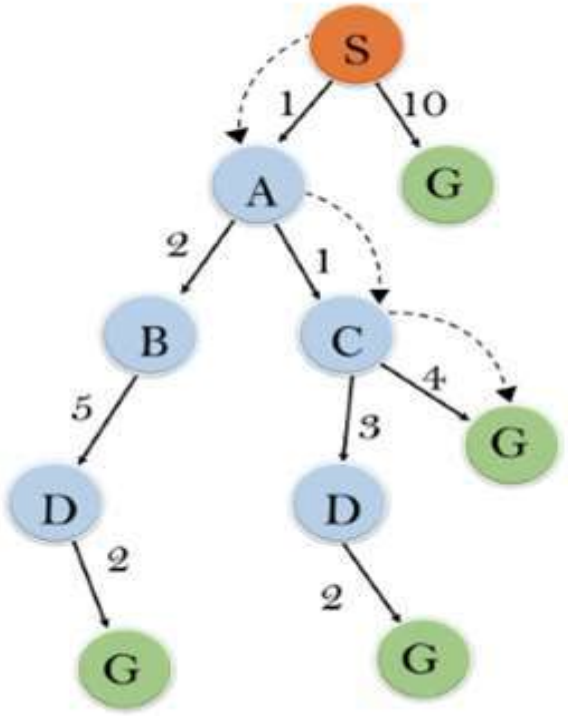
**Iteration1:** {(S--> A, 4), (S-->G, 10)}

**Iteration2:** {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

**Iteration3:** {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

**Iteration 4** will give the final result, as

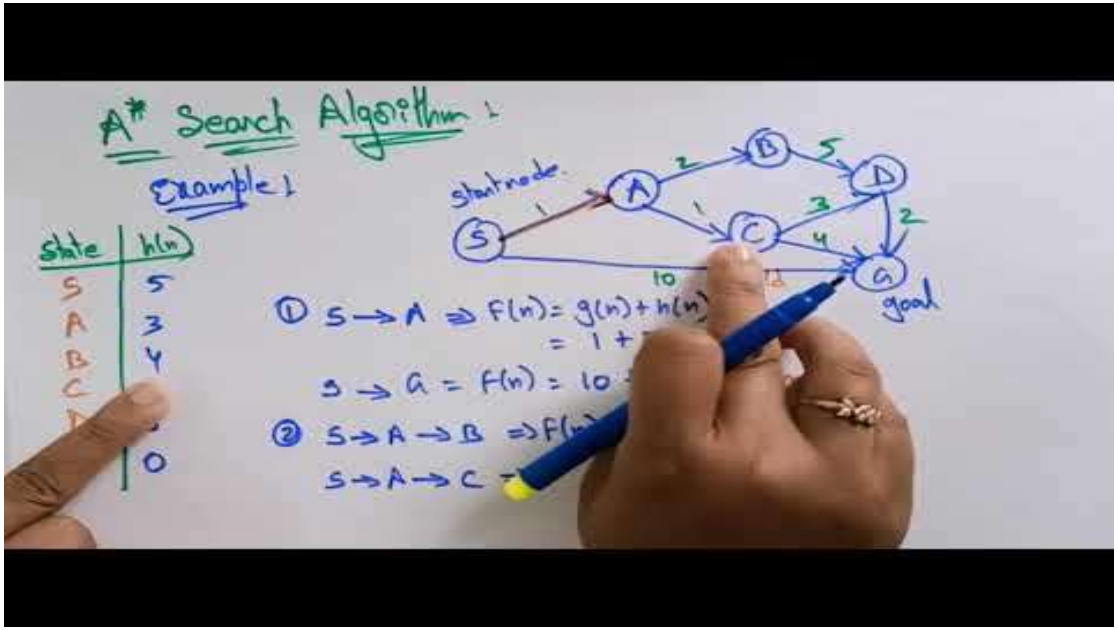
**S--->A--->C--->G** it provides the optimal path with cost 6.

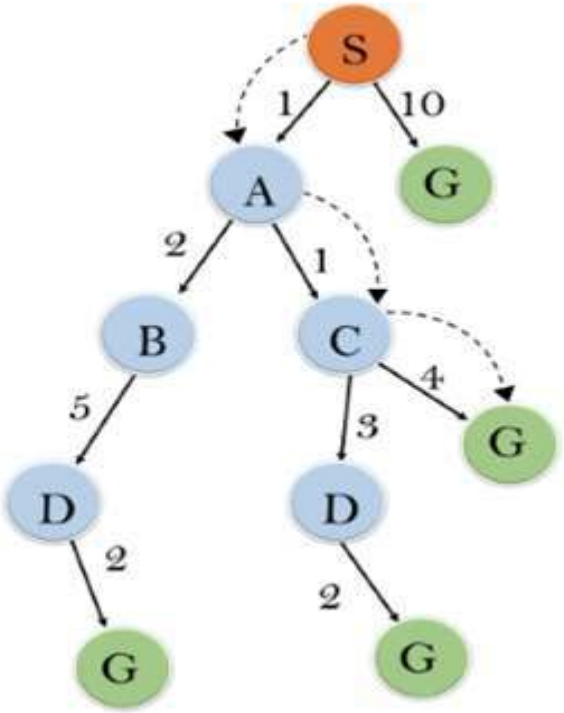



- **Points to remember:**
  - A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
  - The efficiency of A* algorithm depends on the quality of heuristic.
  - A* algorithm expands all nodes which satisfy the condition f(n)

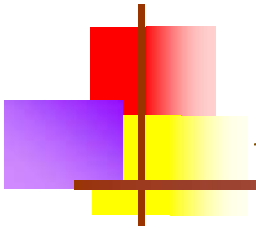
# A* Search Algorithm…

- Solution:



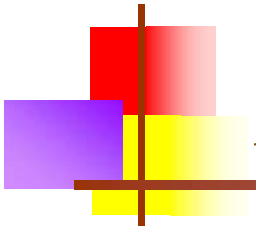




- Source: https://www.youtube.com/watch?v=PzEWHH2v3TE



# A* Search Algorithm…

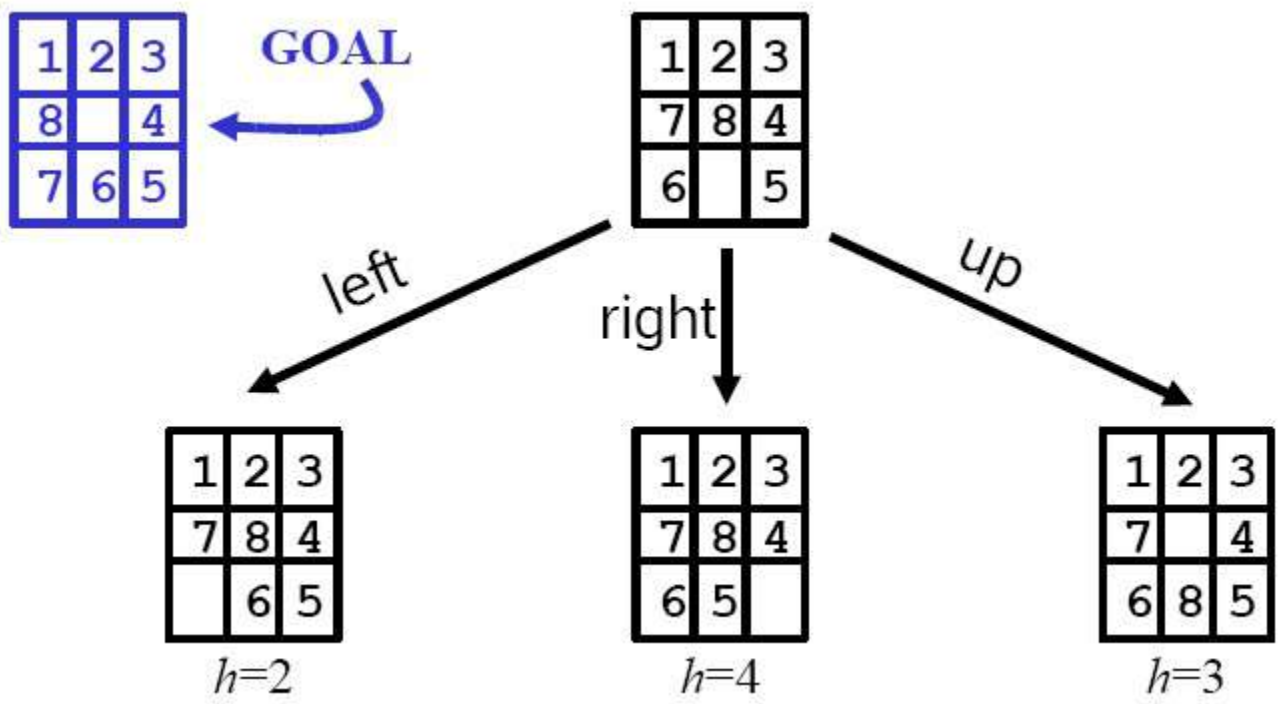
- **Complete:** A* algorithm is complete as long as:
  - Branching factor is finite.
  - Cost at every action is fixed.
- **Optimal:** A* search algorithm is optimal if it follows below two conditions:
  - **Admissible:** the first condition requires for optimality is that h(n) should be an admissible 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)**

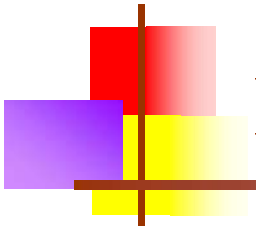


# A* Search Algorithm

- **Advantages:**
  - A* search algorithm is the best algorithm than other search algorithms.
  - A* search algorithm is optimal and complete.
  - This algorithm can solve very complex problems.
- **Disadvantages:**
  - It does not always produce the shortest path as it mostly based on heuristics and approximation.
  - A* search algorithm has some complexity issues.
  - The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

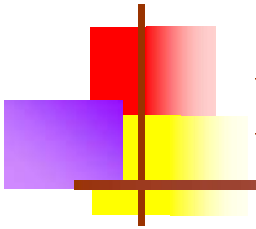
# 8 Puzzle Heuristics





# Hill Climbing Algorithm in AI…

- 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.
- This technique 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. **We don't need to maintain and handle the search tree or graph** as it only keeps a single current state.

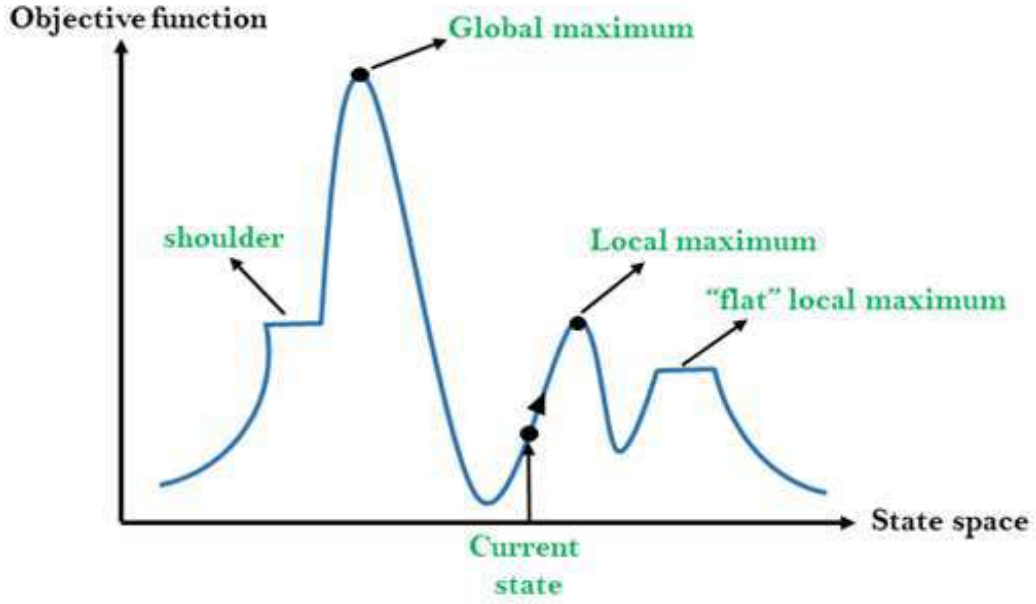


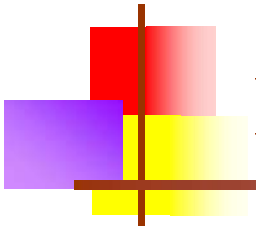
# Hill Climbing Algorithm in AI…

- 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.

# Hill Climbing Algorithm in AI…

- **State-space Diagram for Hill Climbing:**
  - The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.
  - On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.



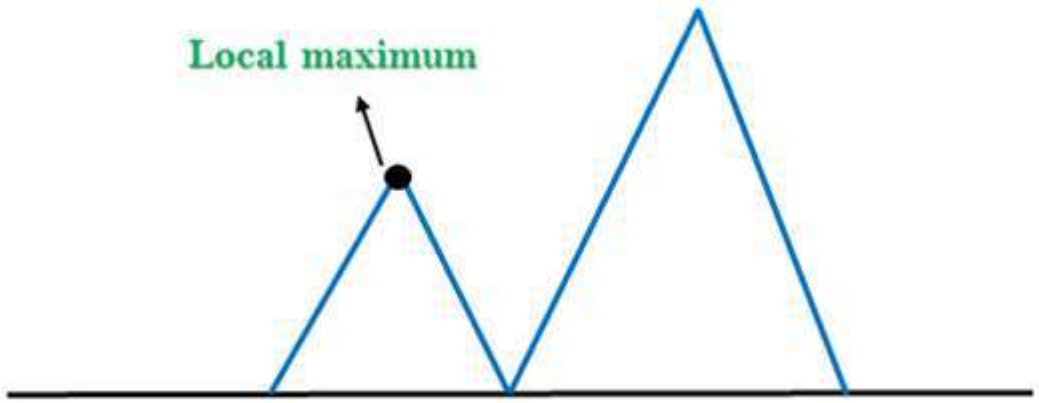


# Hill Climbing Algorithm in AI…

- **Algorithm for Simple Hill Climbing:**
  - **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
  - **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
  - **Step 3:** Select and apply an operator to the current state.
  - **Step 4:** Check new state:
    - If it is goal state, then return success and quit.
    - Else if it is better than the current state then assign new state as a current state.
    - Else if not better than the current state, then return to step2.
  - **Step 5:** Exit..

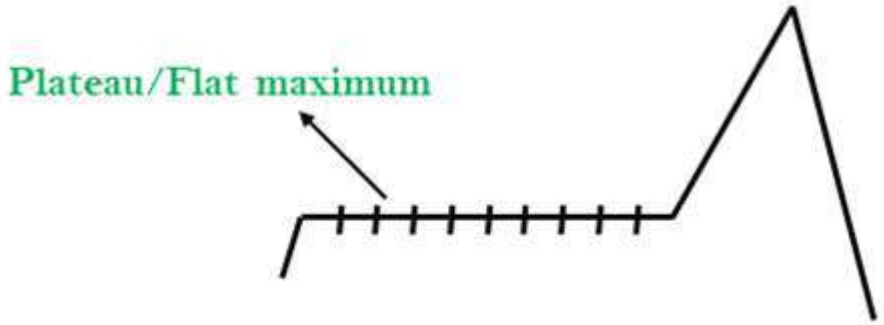
# Hill Climbing Algorithm in AI…

- 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.



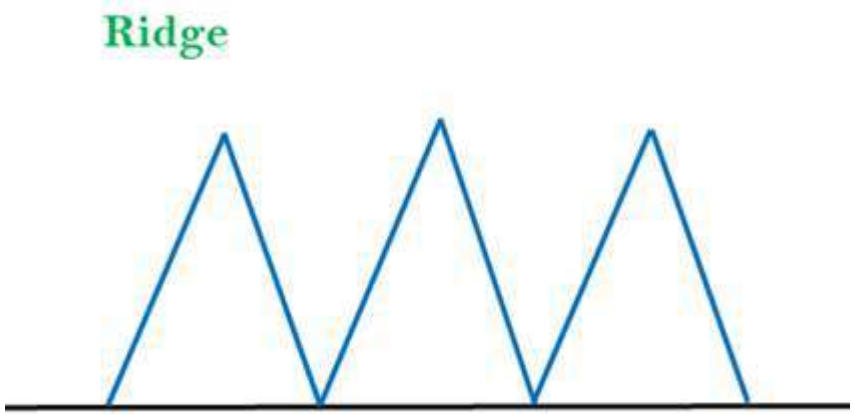
# Hill Climbing Algorithm in AI…

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



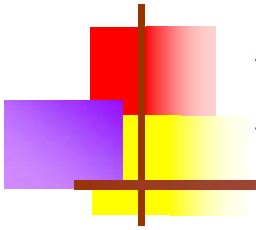
# Hill Climbing Algorithm in AI…

- Problems in Hill Climbing Algorithm:
- **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.



# Hill Climbing Algorithm in AI


- Source: https://www.youtube.com/watch?v=rnTilPSymbE



# Lab Works

- Lab Report

## Implementation of Search Algorithms in AI

[Language Tools: C/C++; Java; Python; Prolog]



# THE END