2.1 LOCAL SEARCH AND OPTIMIZATION

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.

Local search algorithms are widely applied to numerous hard computational problems, including problems from computer science (particularly artificial intelligence), mathematics, operations research, engineering, and bioinformatics. Examples of local search algorithms are WalkSAT, the 2-opt algorithm for the Traveling Salesman Problem and the Metropolis–Hastings algorithm.

A local search algorithm starts from a candidate solution and then iteratively moves to a neighbor solution. This is only possible if a neighborhood relation is defined on the search space. As an example, the neighborhood of a vertex cover is another vertex cover only differing by one node. For boolean satisfiability, neighbors of a truth assignment are usually the truth assignments only differing from it by the evaluation of variable. Same problem may have multiple different neighborhoods defined on it; local optimization with neighborhoods that involve changing up to k components of the solution is often referred to as k-opt.

Typically, every candidate solution has more than one neighbor solution; the choice of which one to move to is taken using only information about the solutions in the neighborhood of the current one, hence the name local search. When the choice of the neighbor solution is done by taking the one locally maximizing the criterion, the metaheuristic takes the name hill climbing. When no improving configurations are present in the neighborhood, local search is stuck at a locally optimal point. This local-optima problem can be cured by using restarts (repeated local search with different initial conditions), or morecomplex schemes based on iterations, like iterated local search, on memory, like reactive search optimization, on memory-less stochastic modifications, like simulated annealing.

Termination of local search can be based on a time bound. Another common choice is to terminate when the best solution found by the algorithm has not been improved in a given number of steps. Local search is an anytime algorithm: it can return a valid solution even if it's interrupted at any time before it ends. Local search algorithms are typically approximation or incomplete algorithms, as the search may stop even if the best solution found by the algorithm is not optimal. This can happen even if termination is due to the impossibility of

improving the solution, as the optimal solution can lie far from the neighborhood of the solutions crossed by the algorithms.

For specific problems it is possible to devise neighborhoods which are very large (exponentially sized). If the best solution within the neighborhood can be found efficiently, such algorithms are referred to as very large-scale neighborhood search algorithms.

Features of Local search

- Keep track of single current state
- Move only to neighboring states
- Ignore paths

Advantages:

- Use very little memory
- Can often find reasonable solutions in large or infinite (continuous) state spaces.
- Features of "Pure optimization" problems
- All states have an objective function
- Goal is to find state with max (or min) objective value
- Does not quite fit into path-cost/goal-state formulation
- Local search can do quite well on these problems.

2.1.1 Hill Climbing Algorithm in Artificial Intelligence

- 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 bestsolution 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.

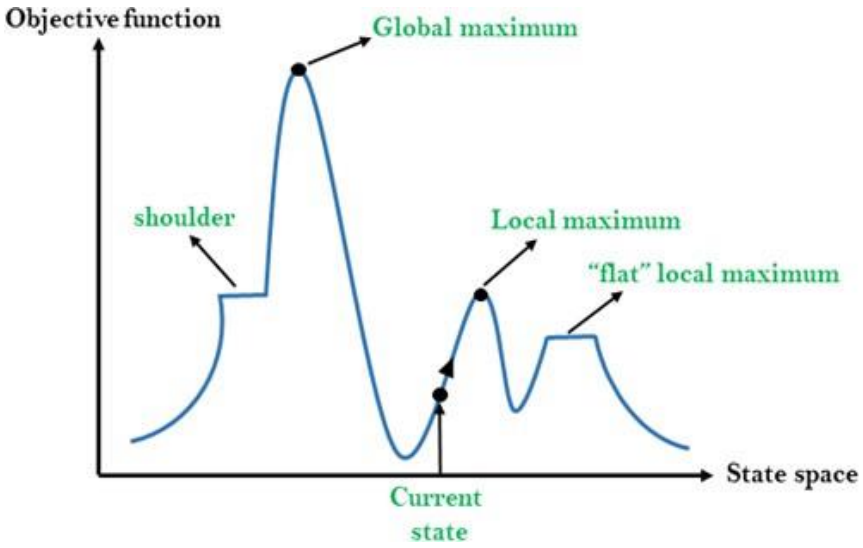
Features of Hill Climbing:

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 decidewhich direction to move in the search space.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing:

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.



Different regions in the state space landscape:

- **Local Maximum:** Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.
- **Global Maximum:** Global maximum is the best possible state of state space landscape. It has the highest value of objective function.
- **Current state:** It is a state in a landscape diagram where an agent is currently present.

- **Flat local maximum:** It is a flat space in the landscape where all the neighbor states of current states have the same value.
- **Shoulder:** It is a plateau region which has an uphill edge.

2.1.2 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

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.

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

Algorithm for Steepest-Ascent hill climbing

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.

- **Step 2:** Loop until a solution is found or the current state does not change.
  - ✓ **Step 3 :** Let SUCC be a state such that any successor of the current state will be better than it.
  - ✓ **Step 4 :** For each operator that applies to the current state:
    - Apply the new operator and generate a new state.
    - Evaluate the new state.
    - If it is goal state, then return it and quit, else compare it to the SUCC.
    - If it is better than SUCC, then set new state as SUCC.
    - If the SUCC is better than the current state, then set current state to SUCC.
- **Step 5:** Exit.

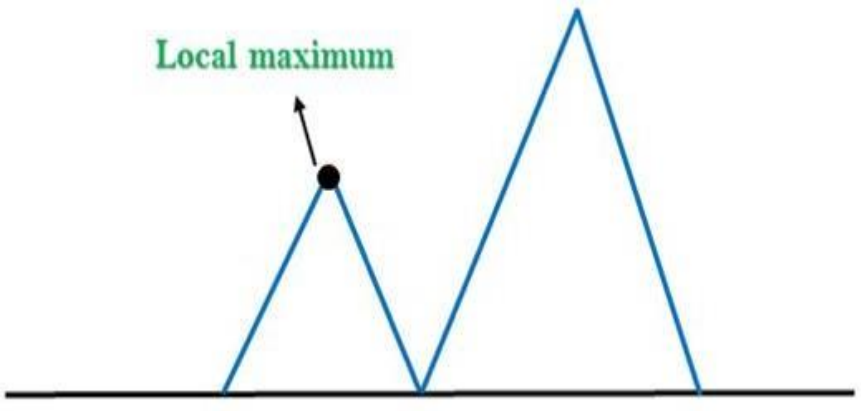
3. Stochastic hill climbing:

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.

Problems in Hill Climbing Algorithm:

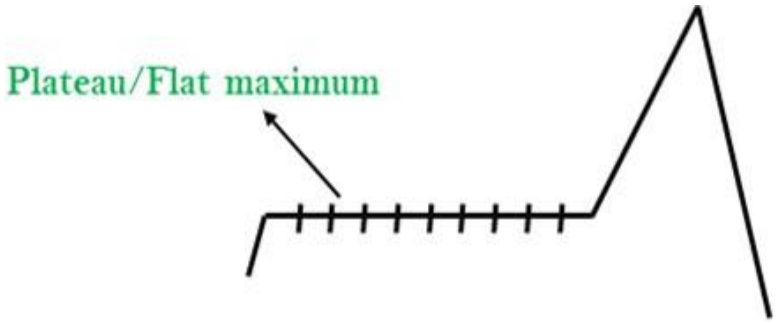
a. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

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.



b. 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.



c. 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.

2.1.3 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.

Annealing is the process of heating and cooling a metal to change its internal structure for modifying its physical properties. When the metal cools, its new structure is seized, and the metal retains its newly obtained properties. In simulated annealing process, the temperature is kept variable. We initially set the temperature high and then allow it to „cool' slowly as the algorithm proceeds. When the temperature is high, the algorithm is allowed to accept worse solutions with high frequency.

- Start
- Initialize k = 0; L = integer number of variables;
- From i → j, search the performance difference Δ.
- If Δ <= 0 then accept else if exp(-Δ/T(k)) > random(0,1) then accept;
- Repeat steps 1 and 2 for L(k) steps.
- k = k + 1;
- Repeat steps 1 through 4 till the criteria is met.
- End

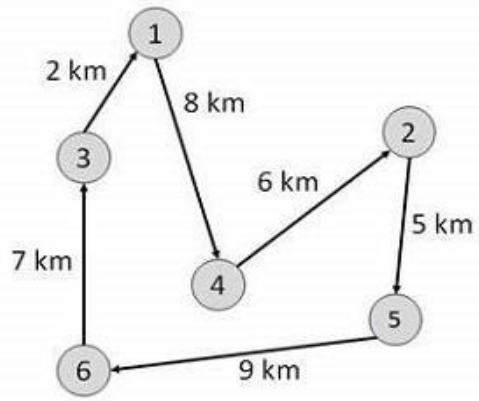
Hill-climbing (Greedy Local Search) max version

function HILL-CLIMBING(problem) return a state that is a local maximum
input: problem, a problem local
variables: current, a node. neighbor, a node.
current ← MAKE-NODE(INITIAL-STATE[problem])
loop do
neighbor ← a highest valued successor of current
if VALUE [neighbor] ≤ VALUE[current] then return STATE[current]
current ← neighbor

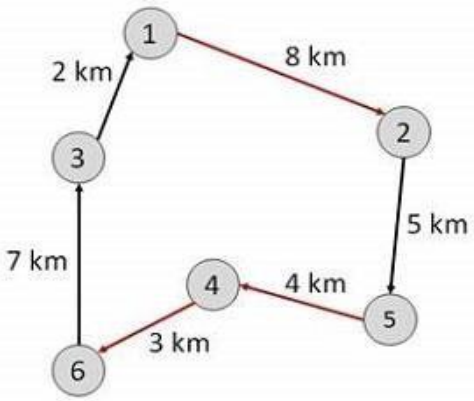
2.1.4 Travelling Salesman Problem

In this algorithm, the objective is to find a low-cost tour that starts from a city, visits all cities en-route exactly once and ends at the same starting city.

Start
Find out all (n -1)! Possible solutions, where n is the total number of cities.
Determine the minimum cost by finding out the cost of each of these (n -1)! solutions.
Finally, keep the one with the minimum cost.
end



Total Distance = 37km



Total Distance = 31km