**Local &Adversarial search**

The search algorithms that we have seen so far are designed to explore search spaces systematically. This systematicity is achieved by keeping one or more paths in memory and by recording which alternatives have been explored at each point along the path. When a goal is found, the *path* to that goal also constitutes a *solution* to the problem. In many problems, however, the path to the goal is irrelevant. For example, in the 8-queens problem, what matters is the final configuration of queens, not the order in which they are added. The same general property holds for many important applications such as integrated-circuit design, factory-floor layout, job-shop scheduling, automatic programming, telecommunications network optimization, vehicle routing, and portfolio management. If the path to the goal does not matter, we might consider a different class of algorithms ones that do not worry about paths at all. Local search algorithms operate using a single current node (rather than multiple paths) and generally move only to neighbours of that node. Typically, the paths followed by the search are not retained. Although local search algorithms are not systematic, they have two key advantages: (1) they use very little memory—usually a constant amount; and (2) they can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable.

In addition to finding goals, local search algorithms are useful for solving pure opimization problems, in which the aim is to find the best state according to an objective function. Many optimization problems do not fit the “standard” search model. For example, nature provides an objective function—reproductive fitness—that Darwinian evolution could be seen as attempting to optimize, but there is no “goal test” and no “path cost” for this problem.

To understand local search, we find it useful to consider the state-space landscape. A landscape has both “location” (defined by the state) and “elevation” (define by the value of the heuristic cost function or objective function). If elevation corresponds to cost, then the aim is to find the lowest valley—a global minimum; if elevation corresponds to an objective function, then the aim is to find the highest peak—a global maximum. (You can convert from one to the other just by inserting a minus sign.) Local search algorithms explore this landscape. A complete local search algorithm always finds a goal if one exists; an optimal algorithm always finds a global minimum/maximum.

**Hill Climbing Search:**

Hill Climbing is heuristic search used for mathematical optimization problems in the field of Artificial Intelligence.

Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum.

* In the above definition, mathematical optimization problems implies that hill climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs. Example[-Travelling salesman problem](https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/) where we need to minimize the distance travelled by salesman.

* ‘Heuristic search’ means that this search algorithm may not find the optimal solution to the problem. However, it will give a good solution in reasonable time.
* A heuristic function is a function that will rank all the possible alternatives at any branching step in search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

Features of Hill Climbing

* 1. Variant of generate and test algorithm: It is a variant of generate and test algorithm. The generated and test algorithm is as follows:

1. *Generate a possible solutions.*
2. *Test to see if this is the expected solution.*
3. *If the solution has been found quit else go to step 1.*

Hence we call Hill climbing as a variant of generate and test algorithm as it takes the feedback from test procedure. Then this feedback is utilized by the generator in deciding the next move in search space.

1. Uses the Greedy approach : At any point in state space, the search moves in that direction only which optimizes the cost of function with the hope of finding the optimal solution at the end.

**Types of Hill Climbing:**

1. **Simple Hill climbing:** It examines the neighbouring nodes one by one and selects the firstneighbouring node which optimizes the current cost as next node.

Algorithm for Simple Hill climbing:

***Step 1:*** *Evaluate the initial state. If it is a goal state then stop and return success.*

*Otherwise, make initial state as current state.*

***Step 2:*** *Loop until the solution state is found or there are no new operators present**which can be applied to current state.*

1. *Select a state that has not been yet applied to the current state and apply it to produce a new state.*
2. *Perform these to evaluate new state*
   * 1. *If the current state is a goal state, then stop and return success.*
   1. *If it is better than the current state, then make it current state and proceed further. iii. If it is not better than the current state, then continue in the loop until a solution is found.*

***Step 3:*** *Exit.*

1. **Steepest-Ascent Hill climbing:** It first examines all the neighboring nodes and thenselects the node closest to the solution state as next node.

***Step 1:*** *Evaluate the initial state. If it is goal state then exit else make the current**state as initial state*

***Step 2:*** *Repeat these steps until a solution is found or current state does not change*

*i. Let ‘target’ be a state such that any successor of the current state will be better than it;*

* + 1. *for each operator that applies to the current state a. apply the new operator and create a new state*
  1. *evaluate the new state*
  2. *if this state is goal state then quit else compare with ‘target’*
  3. *if this state is better than ‘target’, set this state as ‘target’*
  4. *if target is better than current state set current state to Target* ***Step 3:*** *Exit*

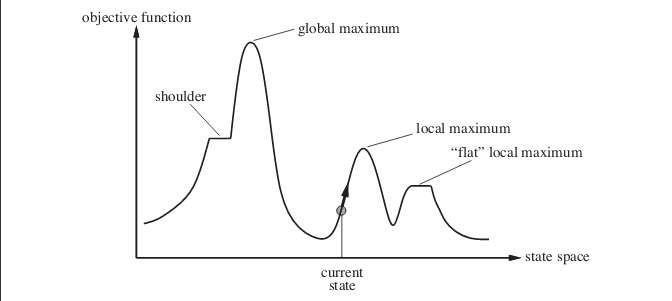
1. **Stochastic hill climbing :** It does not examine all the neighbouring nodes beforedeciding which node to select .It just selects a neighbouring node at random, and decides (based on the amount of improvement in that neighbour) whether to move to that neighbour or to examine another.

**State Space diagram for Hill Climbing**

State space diagram is a graphical representation of the set of states our search algorithm can reach vs the value of our objective function (the function which we wish to maximize). X-axis: denotes the state space ie states or configuration our algorithm may reach.

Y-axis: denotes the values of objective function corresponding to to a particular state.

The best solution will be that state space where objective function has maximum value (global maximum).



Different regions in the State Space Diagram

1. Local maximum: It is a state which is better than its neighbouring state however there exists a state which is better than it(global maximum). This state is better because here value of objective function is higher than its neighbours.
2. Global maximum: It is the best possible state in the state space diagram. This because at this state, objective function has highest value.
3. Plateua/flat local maximum: It is a flat region of state space where neighbouring states have the same value.
4. Ridge: It is region which is higher than its neighbours but itself has a slope. It is a special kind of local maximum.
5. Current state: The region of state space diagram where we are currently present during the search.
6. Shoulder: It is a plateau that has an uphill edge.

Problems in different regions in Hill climbing

Hill climbing cannot reach the optimal/best state (global maximum) if it enters any of the following regions:

1. **Local maximum**: At a local maximum all neighbouring states have a values whichis worse than than the current state. Since hill climbing uses greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.

To overcome local maximum problem: Utilize backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.

1. **Plateau:** On plateau all neighbors have same value. Hence, it is not possible toselect the best direction.

**To overcome plateaus:** Make a big jump. Randomly select a state far away from current state. Chances are that we will land at a non-plateau region

Ridge: Any point on a ridge can look like peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state. To overcome

Ridge: In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

**Simulated Annealing**

Simulated Annealing is a variation on hill climbing. Simulated annealing technique can be explained by an analogy to annealing in solids. In the annealing process in case of solids, a solid is heated past melting point and then cooled.

With the changing rate of cooling, the solid changes it’s properties. If the liquid is cooled slowly, it gets transformed in steady frozen state and forms crystals. While, if it is cooled quickly, the crystal formation will not get enough time and it produces imperfect crystals.

The aim of physical annealing process is to produce a minimal energy final state after raising the substance to high energy level. Hence in simulated annealing we are actually going downhill and the heuristic function is a minimal heuristic. The final state is the one with minimum value, and rather than climbing up in this case we are descending the valley.

The idea is to use simulated annealing to search for feasible solutions and converge to an optimal solution. In order to achieve that, at the beginning of the process, some downhill moves may made. These downhill moves are made purposely, to do enough exploration of the whole space early on, so that the final solution is relatively insensitive to the starting state. It reduces the chances of getting caught at a local maximum, or plateau, or a ridge.

**Algorithm:**

*Evaluate the initial state.*

*Loop until a solution is found or there are no new operators left to be applied:*

* *Set T according to an annealing schedule.*
* *Select and apply a new operator.*
* *Evaluate the new state:*

*Goal-> quit*

*∆E=Val (current state)-Val (new state)*

*∆E<0->new current state*

*else->new current state with probability-∆E/kT*

* We observe in the algorithm that, if the next state is better than the current, it readily accepts it as a new current state. But in case when the next state is not having the desirable value even then it accepts that state with some probability. Where E is the positive change in the energy level, T is temperature and k is Boltzmann's constant.
* Thus in simulated annealing there are very less chances of large uphill moves than the small one. Also, the probability of uphill moves decreases with the temperature decrease. Hence uphill moves are likely in the beginning of the annealing process, when the temperature is high. As the cooling process starts, temperature comes down, in turn the uphill moves. Downhill moves are allowed any time in the whole process. In this way, comparatively very small upward moves are allowed till finally, the process converges to a local minimum configuration, ie the desired low point destination in the valley.

**Local Beam Search**

Keeping just one node in memory might seem to be an extreme reaction to the problem of memory limitations. The local beam search algorithm keeps track of *k* states rather than just one. It begins with k randomly generated states. At each step, all the successors of all *k* states are generated. If anyone is a goal, the algorithm halts. Otherwise, it selects the *k* best successors from the complete list and repeats.

In this algorithm, it holds k number of states at any given time. At the start, these states are generated randomly. The successors of these k states are computed with the help of objective function. If any of these successors is the maximum value of the objective function, then the algorithm stops.

Otherwise the (initial k states and k number of successors of the states = 2k) states are placed in a pool. The pool is then sorted numerically. The highest k states are selected as new initial states. This process continues until a maximum value is reached.

**Algorithm**

function BeamSearch( problem, k), returns a solution state.

start with k randomly generated states

loop

generate all successors of all k states

if any of the states = solution, then return the state else select the k best successors

end