Informed Search :

Systematic Method for Problem Solving

The *systematic problem-solving method* that guarantees a solution, provided that one exists, is known as the algorithmic *method* or algorithm *problem-solving.* This *process* involves a *systematic* and structured *approach* to identifying, analyzing, and resolving problems.

Six step guide to help you solve problems

Step 1: Identify and define the problem

- State the problem as clearly as possible.

Step 2: Generate possible solutions

- List all the possible solutions; don't worry about the quality of the solutions at this stage.

Step 3: Evaluate alternatives

- The next step is to go through and eliminate less desirable or unreasonable solutions.
- Order the remaining solutions in order of preference.
- Evaluate the remaining solutions in terms of their advantages and disadvantages.

Step 4: Decide on a solution

- Specify who will take action.
- Specify how the solution will be implemented.
- Specify when the solution will be implemented. For example: tomorrow morning, phone the gas company and negotiate to pay the gas bill next month.

Step 5: Implement the solution

- Implement the solution as planned.

Step 6: Evaluate the outcome

- Evaluate how effective the solution was.

Generate and Test method for Problem Solving

Algorithm: Generate-And-Test

Step-1. Generate a possible solution.

Step-2. Test to see if this is the expected solution.

Step-3. If the solution has been found quit else go
to Step 1.

This approach is what is known as British Museum algorithm: finding an object in the British Museum by wandering randomly.

- Generate-and-test: A brute-force depth first search in its simplest form
- Hill-climbing: a variation on generate and test approach
- Backtracking is a simple universal strategy, that requires the algorithms to maintain state information.

Backtracking:

Basic idea:

1.) Try a rule
2.) If no solution, go back to the state at 1, and try another rule

• Backtracking can avoid local minima
• Backtracking allows alternative paths to be explored.
• If rule selection is guided by information then the backtracking can be more efficient.

BACK TRACKING IN CSP: The idea backtracking search is to repeat the following steps

- pick an unassigned variable
- assign it a value that doesn't violate any constraints
  - if there is no such value, then backtrack to the previous step and choose a new variable

A basic backtracking search is complete, i.e. it will find solutions if they exist, but in practice it is often very slow on large problems

- Depth-first search for CSPs with single-variable assignments is called backtracking search.
- Backtracking search is a special kind of Depth-first search, but they are not the same.
- Backtracking search is the basic uninformed algorithm for CSPs

Backtracking is a systematic method to iterate through all the possible configurations of a search space. It is a general algorithm/technique which must be customized for each individual application.

Backtracking is often faster than the brute force approach as it does not require **generating and testing** all possible solutions

Generate and Test is the simplest of the algorithms to identify a solution for a CSP. In this algorithm, each of the possible solutions is attempted (each value enumerated for each variable) and then tested for the solution.

Local Search Algorithms:

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) until a solution deemed optimal is found or a time bound is elapsed.

Local Search Algorithms:

- Keep track of single current state
- Move only to neighboring states
- Ignore paths (No memory or very less memory)

1) Hill Climbing (greedy hill climbing)
2) Genetic Algorithm

Hill Climbing:

Hill climbing search algorithm is simply a loop that continuously moves in the direction of increasing value. It stops when it reaches a "peak" where no neighbour has higher value. This algorithm is considered to be one of the simplest procedures for implementing heuristic search. The hill climbing comes from that idea if you are trying to find the top of the hill and you go up direction from where ever you are.

The hill climbing is a variant of generate and test in which direction the search should proceed. At each point in the search path, a successor node that appears to reach for exploration.

Algorithm: Hill Climbing

Step 1: Evaluate the starting state. If it is a goal state, then stop and return success.
Step 2: Else, continue with the starting state as considering it as a current state.
Step 3: Continue step-4 until a solution is found i.e. until there are no new states left to be applied in the current state.
Step 4:
a. Select a state that has not been yet applied to the current state and apply it to produce a new state.
b. Procedure to evaluate a new state.
    i. If the current state is a goal state, then stop and return success.
    ii. 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 5: Exit.

- 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.

Hill Climbing Features:

- 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.

A major problem of hill-climbing strategies is their tendency to become stuck at local maxima

In hill climbing, basically, there are three main problems.
• Local maxima
• Plateau
• Ridges

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:

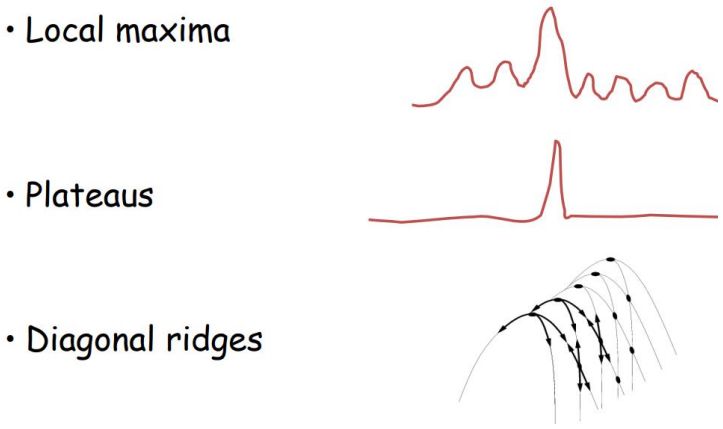
(i) Random restart: keep restarting search from random locations until a goal is found
(ii) Problem reformulation: reformulate search space to eliminate these problematic features

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.

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.

Solutions: With the use of bidirectional search, or by moving in different directions, we can improve this problem



General Solution to Hill Climbing :

- Greedy: move to the neighbor with largest value
- Random Walk: move to a random neighbour
- Random Restart: Resample a new current state

Uniform Cost Search: Like BFS, but for variable-cost actions

- Frontier is a priority queue, sorted by **g(n)**
- Finds the optimal path

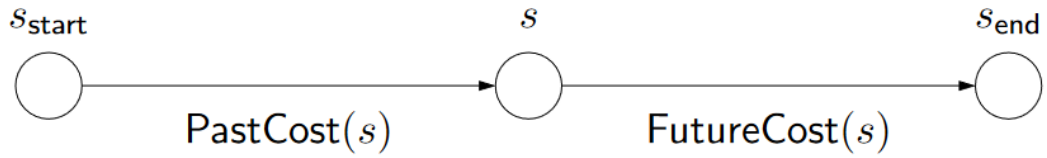
Greedy Search:

- Frontier is a priority queue, sorted by **h(n)**
- Not optimal. Not even complete.

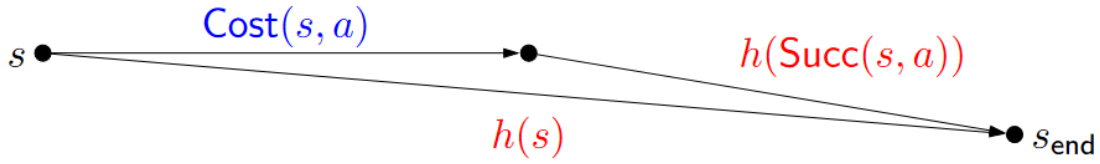
A* Search:

- Frontier is a priority queue, sorted by **f(n)=g(n)+h(n)**
- Optimal and complete, as long as **h(n)** is admissible

The only *difference between tree search and graph search* is that tree search does *not need store the explored set*, because we are guaranteed never to attempt to visit the same state twice.



UCS: explore states in order of *PastCost(s)*
A*: explore in order of *PastCost(s) + h(s)*
A heuristic h(s) is estimated *FutureCost(s)*
Uniform-cost orders by path cost, or *backward cost g(n)*
Greedy BFS orders by goal proximity, or *forward cost h(n)*



A heuristic *h(s)* is admissible if *h(s) ≤ FutureCost(s)*
If a heuristic *h(s)* is consistent, then *h(s)* is admissible.
A* is admissible if it uses an admissible heuristic, and *h(goal) = 0*
Greedy Best-First Search:
Evaluation function f(n) = h(n) (heuristic)
= estimate of cost from n to goal