LP1

AIR MINI PROJECT REPORT

Submitted by : Shubham Fargade(41327)
Atul Kardak(41340)
Yash Kasat (41441)
Hardik Chandak(41331)
Batch : Q3

1. Title:

Implement Hill-climbing algorithm.

2. Objective:

To understand the Hill climbing algorithm algorithm using Heuristic search technique.

3. Problem Statement:

Heuristic search technique to implement Hill-climbing algorithm.

4. Outcomes:

The students will be able to understand the various heuristic search techniques for Hill climbing algorithms.

5. Software and hardware requirement:

1. 32/64-bit Open source Linux/windows
2. Python3 installed

6. Theory and concept:

6.1 Hill Climbing

Hill Climbing is a 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 problem 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 where we need to minimize the distance traveled by the 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 a reasonable time**.**

- A heuristic function is a function that will rank all the possible alternatives at any branching step in a search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

6.2 Features of Hill Climbing

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

   1. Generate 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 the test procedure. Then this feedback is utilized by the generator in deciding the next move in search space.
2. 2.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.

7. Algorithm

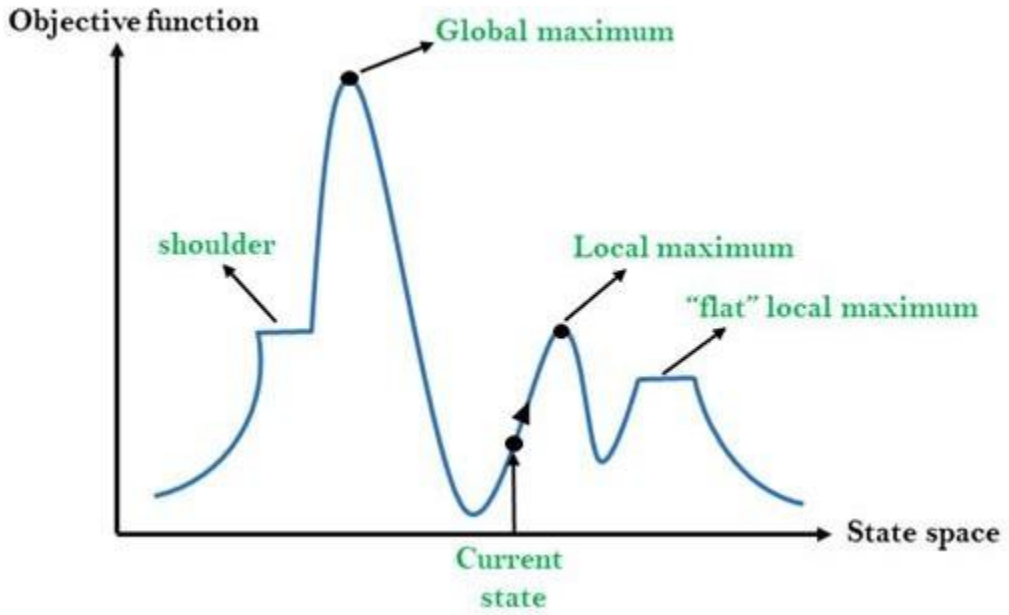
1. ***Step 1 :*** *Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.*
2. ***Step 2 :*** *Loop until the solution state is found or there are no new operators present which can be applied to the current state.*
3. *a) Select a state that has not been yet applied to the current state and apply it to produce a new state.*
4. *b) Perform these to evaluate new state*
5. *i. If the current state is a goal state, then stop and return success.*
6. *ii. If it is better than the current state, then make it current and proceed further.*
7. *iii. If it is not better than the current state, then continue in the loop until a solution is found.*
8. ***Step 3 :*** *Exit.*

8. History and Applications

A Heuristic is a technique to solve a problem faster than classic methods, or to find an approximate solution when classic methods cannot. This is a kind of a shortcut as we often trade one of optimality, completeness, accuracy, or precision for speed. A Heuristic (or a heuristic function) takes a look at search algorithms. At each branching step, it evaluates the available information and makes a decision on which branch to follow. It does so by ranking alternatives. The Heuristic is any device that is often effective but will not guarantee work in every case.

So *why do we need heuristics*? One reason is to produce, in a reasonable amount of time, a solution that is good enough for the problem in question. It doesn't have to be the best- an approximate solution will do since this is fast enough. Most problems are exponential. Heuristic Search let us reduce this to a rather polynomial number. We use this in AI because we can put it to use in situations where we can't find known algorithms.

Hill Climbing technique can be used to solve many problems, where the current state allows for an accurate evaluation function, such as Network-Flow, Travelling Salesman problem, 8-Queens problem, Integrated Circuit design, etc.Hill Climbing is used in inductive learning methods too.



9.Program

```
import string
import random

def score(list1,list2):
    score_gen = 0
    for i in range(len(list1)):
        if list1[i] == list2[i]:
            score_gen = score_gen+1
    return score_gen / len(list1)

def create(clist,list1,list2):
    for i in range(len(list1)):
        if list1[i] == list2[i]:
            clist[i] = list1[i]
    return clist

def random_generator(goal):
#    characters = string.ascii_lowercase+" "
 # randstring = ""
  # for i in range(len(goal)):
   #     randstring = randstring + characters[random.randrange(len(characters))]
   # randList = [randstring[i] for i in range(len(randstring))]
    randList = [random.choice(string.printable) for _ in range(len(goal))]
    return randList

def main():
    goal = "my name is Yash"
    goallist = [goal[i] for i in range(len(goal))]
    clist = [' ' for i in range(len(goal))]
    random_string = random_generator(goallist)
    clist = create(clist,random_string,goallist)
    score_gen = score(clist,goallist)
    iteration = 0
    while (score_gen < 1):
        random_string = random_generator(goallist)
        clist = create(clist,random_string,goallist)
        score_gen = score(clist,goallist)
        iteration = iteration+1
        print(score_gen," : ","".join(clist))
    print("Total iterations: ",iteration)

main()
```

```
import random
import string
```

```
def generate_random_solution(length=13):
    return [random.choice(string.printable) for _ in range(length)]

def evaluate(solution):
    target = list("Hello, World!")
    diff = 0
    for i in range(len(target)):
        s = solution[i]
        t = target[i]
        diff += abs(ord(s) - ord(t))
    return diff

def mutate_solution(solution):
    index = random.randint(0, len(solution) - 1)
    solution[index] = random.choice(string.printable)

best = generate_random_solution()
best_score = evaluate(best)

while True:
    print('Best score so far', best_score, 'Solution', "".join(best))

    if best_score == 0:
        break

    new_solution = list(best)
    mutate_solution(new_solution)

    score = evaluate(new_solution)
    if evaluate(new_solution) < best_score:
        best = new_solution
        best_score = score
```

10. Conclusion

Successfully implemented Hill-Climbing Algorithm using heuristic search in python.