Experiment 4

Shashwat Shah

60004220126

Div B   C2-2

Aim : To implement local search algorithm: Hill climbing.

Theory : Hill climbing is a local search algorithm in AI to find the best possible solution to a problem. It is named after the idea that it climbs the hills of search space to find the most optimal solution but it can have limitations and may not be always find the global optimum. Current state begins with an initial state and explores the best neighbour iteratively to determine its next move.

Objective function - To evaluate the quality of a state, the goal is to maximize or minimize this function. It guides the searching towards better solutions

Advantages :-

Simplicity - It is straight forward to implement and understand.

Efficiency - It can quickly find local optimum due to its simple structure

Disadvantages.

i) Limited Exploration - It only considers the best immediate neighbours, hence entire graph cannot be explored.

ii) local optimum - It is prone to getting stuck in the local optimum and can miss the global optimum

iii) Dependency on initial state.

iv) It does not maintain a history of visited states.

Conclusion : Hence, we have implemented the local search algorithm, Hill climbing.

Experiment 4

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Aim:

Program to implement Local Search algorithm: Hill climbing search.

Theory:

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 neighbour has a higher value. This 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 travelled by the salesman. It is also called greedy local search as it only looks to its good immediate neighbour state and not beyond that. In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state. A node of hill climbing algorithm has two components which are state and value. It is mostly used when a good heuristic is available.

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.

Types of Hill Climbing Algorithm are:

1. Simple hill Climbing
2. Steepest-Ascent hill-climbing
3. Stochastic hill Climbing

Algorithm

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 the current state.

     a) Select a state that has not been yet applied to the current state and apply it to produce a new state.

     b) Perform these to evaluate new state:

     1. If the current state is a goal state, then stop and return success.

     2. If it is better than the current state, then make it current state and proceed further.

     3. If it is not better than the current state, then continue in the loop until a solution is found.

Step 3: Exit.

Code:

```
import copy visited_states
= [] def
gn(curr_state,prev_heu,go
al_state):    global
visited_states    state =
```

```python
        copy.deepcopy(curr_state
    )    for i in
range(len(state)):
        temp = copy.deepcopy(state)
        if len(temp[i]) > 0:            elem =
temp[i].pop()            for j in
range(len(temp)):
            temp1 = copy.deepcopy(temp)
            if j != i:
                temp1[j] = temp1[j] + [elem]
                if (temp1 not in visited_states):
                    curr_heu=heuristic(temp1,goal_state)
                    if curr_heu>prev_heu:
                        child = copy.deepcopy(temp1)
                        return child

    return 0

def heuristic(curr_state,goal_state):
    goal_=goal_state[3]    val=0    for
i in range(len(curr_state)):
    check_val=curr_state[i]        if
len(check_val)>0:            for j in
range(len(check_val)):            if
check_val[j]!=goal_[j]:

            # val-=1
val-=j            else:
```

```python
   # val+=1
   val+=j
   return val

def sln(init_state,goal_state):
   global visited_states
   if (init_state == goal_state):
      print (goal_state)
      print("solution found!")
      return

   current_state = copy.deepcopy(init_state)

   while(True):
      visited_states.append(copy.deepcopy(current_state))
      print(current_state)
      prev_heu=heuristic(current_state,goal_state)
      child = gn(current_state,prev_heu,goal_state)
      if child==0:
         print("Final state - ",current_state)
         return

      current_state = copy.deepcopy(child)

def hc():
   global visited_states
   initial = [[],[],[],['B','C','D','A']]
   goal = [[],[],[],['A','B','C','D']]
   sln(initial,goal)
```

hc()

```

[[], [], [], ['B', 'C', 'D', 'A']]
[['A'], [], [], ['B', 'C', 'D']]
[['A', 'D'], [], [], ['B', 'C']]
[['A'], ['D'], [], ['B', 'C']]
[['A'], ['D'], ['C'], ['B']]
[['A', 'B'], ['D'], ['C'], []]
[['A', 'B', 'C'], ['D'], [], []]
[['A', 'B', 'C', 'D'], [], [], []]
Final state -  [['A', 'B', 'C', 'D'], [], [], []]


Process Finished.
>>>
```

Conclusion:

Hill Climbing algorithm is good in solving the optimization problem while using only limited computation power.