**Experiment:6**

**Aim: Write a Program to** **Implement Hill Climbing Method.**

Hill climbing is a local search algorithm used for optimization problems. It starts with an initial solution and iteratively makes small improvements to the current solution until it reaches an optimal solution or a local maximum.

Here's a step-by-step explanation of the Hill Climbing method:

1. Initialization: Start with an initial solution. This solution could be randomly generated or based on some heuristic.
2. Evaluation: Evaluate the current solution by calculating its objective function value. The objective function represents the measure of the solution's quality that needs to be optimized.
3. Neighborhood Generation: Generate neighboring solutions by making small modifications or changes to the current solution. These modifications could involve changing a single component of the solution at a time or making more complex alterations.
4. Selection of the Next Solution: Choose the best neighboring solution based on its objective function value. If the objective function value of the neighboring solution is better than the current solution, move to that solution. Otherwise, terminate the algorithm or consider alternative strategies (such as random restarts or simulated annealing).
5. Termination Condition: Determine the termination condition for the algorithm. This condition could be reaching a maximum number of iterations, achieving a certain objective function value, or stagnation (where the algorithm fails to find better solutions after several iterations).
6. Iteration: Repeat steps 3-5 until the termination condition is met.

**Implement Hill Climbing Method**

import random

# Define the objective function to be optimized

def objective\_function(x):

return x\*\*2

# Define the hill climbing function

def hill\_climbing(objective\_function, x\_start, step\_size, max\_iterations):

x\_current = x\_start

for i in range(max\_iterations):

# Generate a random neighbor within the specified step size

x\_neighbor = x\_current + random.uniform(-step\_size, step\_size)

# Evaluate the objective function for the current and neighbor solutions

current\_value = objective\_function(x\_current)

neighbor\_value = objective\_function(x\_neighbor)

# Move to the neighbor if it has a better objective value

if neighbor\_value < current\_value:

x\_current = x\_neighbor

return x\_current, objective\_function(x\_current)

# Define the initial starting point, step size, and maximum number of iterations

x\_start = 10

step\_size = 0.1

max\_iterations = 1000

# Perform hill climbing optimization

best\_solution, best\_value = hill\_climbing(objective\_function, x\_start, step\_size, max\_iterations)

print("Best solution found:", best\_solution)

print("Objective value at best solution:", best\_value)

In this implementation:

* We define an objective function **objective\_function(x)** that we want to minimize. Here, it's a simple quadratic function **x\*\*2**, but you can replace it with any other function according to your optimization problem.
* The **hill\_climbing** function iteratively generates neighboring solutions by perturbing the current solution (**x\_current**) within a specified step size. It accepts parameters like the objective function, initial starting point (**x\_start**), step size (**step\_size**), and maximum number of iterations (**max\_iterations**).
* After reaching the termination condition (maximum iterations), the function returns the best solution found along with its objective value.
* Finally, we print out the best solution and its objective value.

**Output:**

Best solution found: 0.0001596137510846557

Objective value at best solution: 2.547654953531443e-08

**Date of experiment performed:**

**Day of experiment performed:**

**Date of experiment submission:**

**Day of experiment Submission:**

Faculty Co-ordinator Signature