Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2025-26

Class:	**TE**	**Semester:**	**V**
Course Code:	**CSL502**	**Course Name:**	**Artificial Intelligence**

Name of Student:	**Pranita Kumbhar**
Roll No. :	**70**
Experiment No.:	**05**
Title of the Experiment:	**Implement Hill Climbing Search for problem solving**
Date of Performance:	**25/08/2025**
Date of Submission:	**01/08/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mrs. Rujuta Vartak

Signature :

Date:

Aim: Implement Hill Climbing Search for problem solving.

Objectives:

Understand the concept of Hill Climbing as a local search algorithm**.**

Theory:

Hill Climbing is a heuristic local search algorithm used for mathematical optimization and artificial intelligence problems. It iteratively moves to neighboring states that improve the evaluation function (heuristic) value and halts when no further improvements can be found.

It works on the principle of greedy ascent, always choosing the best local move based on a heuristic function. However, it can get stuck in:

- Local maxima: A state better than neighbors but not optimal.
- Plateaus: A flat region where all neighboring states have the same value.
- Ridges: A narrow path to the optimal state that requires indirect moves.

Variants of Hill Climbing:

1. Simple Hill Climbing: Considers one neighbor at a time.
2. Steepest-Ascent Hill Climbing: Evaluates all neighbors and chooses the best.
3. Stochastic Hill Climbing: Randomly selects one of the better neighbors.
4. Random Restart Hill Climbing: Runs the algorithm multiple times with different start states.

Pseudocode: Hill Climbing Search

```
HILL-CLIMBING(problem):
  current ← INITIAL-STATE(problem)
    loop do:
     neighbors ← EXPAND(current)
     if neighbors is empty:
        return current   // No more moves possible
     next ← a neighbor with the highest VALUE among neighbors
```

```
if VALUE(next) ≤ VALUE(current):
    return current   // No improvement, return current state
    current ← next       // Move to the better neighbor
```

Explanation of Terms:

- INITIAL-STATE(problem): Starting state of the problem.
- EXPAND(current): Returns all possible neighboring states from the current state.
- VALUE(state): Heuristic function to evaluate the quality of a state.
- The loop continues as long as a better neighbor is found.
- It **stops** when no neighbor is better than the current state (i.e., local maximum or plateau).

Advantages:

- Space-efficient (uses constant memory).
- Fast for simple problems.

Disadvantages:

- Can get stuck in:
    - **Local Maxima:** A peak that is lower than the global maximum.
    - **Plateaus:** Flat areas where all neighbors have the same value.
    - **Ridges:** Paths that require indirect moves to reach better states.

Python Code Implementation:

```
import random
import numpy as np

# Objective function (maximize this function)
def objective(x):
    return -x[0] ** 2 + 5  # Maximum at x = 0, f(x) = 5

# Generate neighboring states
def generate_neighbors(x, step_size=0.1):
    return [np.array([x[0] + step_size]), np.array([x[0] - step_size])]



# Hill Climbing algorithm
def hill_climbing(objective, initial, n_iterations=100, step_size=0.1):
    current = np.array([initial])
    current_eval = objective(current)

    for i in range(n_iterations):
        neighbors = generate_neighbors(current, step_size)
        neighbor_evals = [objective(n) for n in neighbors]

        best_idx = np.argmax(neighbor_evals)
        if neighbor_evals[best_idx] > current_eval:
            current = neighbors[best_idx]
            current_eval = neighbor_evals[best_idx]
            print(f"Step {i+1}: x = {current[0]:.4f}, f(x) = {current_eval:.4f}")
        else:
            print("No better neighbors found. Algorithm converged.")
            break

    return current, current_eval

# Initial guess
initial_guess = 2.0

# Run the Hill Climbing algorithm
solution, value = hill_climbing(objective, initial_guess, n_iterations=100, step_size=0.1)

print(f"\nBest solution x = {solution[0]:.4f}, f(x) = {value:.4f}")
```

Output :

```
Output
Step 1: x = 1.9000, f(x) = 1.3900
Step 2: x = 1.8000, f(x) = 1.7600
Step 3: x = 1.7000, f(x) = 2.1100
Step 4: x = 1.6000, f(x) = 2.4400
Step 5: x = 1.5000, f(x) = 2.7500
Step 6: x = 1.4000, f(x) = 3.0400
Step 7: x = 1.3000, f(x) = 3.3100
Step 8: x = 1.2000, f(x) = 3.5600
Step 9: x = 1.1000, f(x) = 3.7900
Step 10: x = 1.0000, f(x) = 4.0000
Step 11: x = 0.9000, f(x) = 4.1900
Step 12: x = 0.8000, f(x) = 4.3600
Step 13: x = 0.7000, f(x) = 4.5100
Step 14: x = 0.6000, f(x) = 4.6400
Step 15: x = 0.5000, f(x) = 4.7500
Step 16: x = 0.4000, f(x) = 4.8400
Step 17: x = 0.3000, f(x) = 4.9100
Step 18: x = 0.2000, f(x) = 4.9600
Step 19: x = 0.1000, f(x) = 4.9900
Step 20: x = -0.0000, f(x) = 5.0000
No better neighbors found. Algorithm converged.

Best solution x = -0.0000, f(x) = 5.0000
```

Conclusion:

1. **Comment on your implemented Program and result you got.**

The implemented Hill Climbing program successfully demonstrated how a local search algorithm can find an optimal solution through iterative improvement. The algorithm started from an initial random state and progressively moved toward better neighboring states based on the heuristic (objective) function.

For the given function f(x)=−x2+5f(x) = -x^2 + 5f(x)=−x2+5, the algorithm correctly converged to the optimal solution at x = 0, with the maximum value f(x) = 5. This confirms that the implementation works efficiently for problems with a smooth and unimodal search space.

2. **What is the main drawback of the Hill Climbing algorithm, and how can it be overcome?**

The main drawback of the Hill Climbing algorithm is that it can easily get stuck in local maxima, plateaus, or ridges, where no better neighboring state is available even though the global optimum existselsewhere.
Since the algorithm only moves toward immediate improvements, it lacks the ability to backtrack or explore beyond local peaks.

To overcome this limitation, several strategies can be used:

1. Random Restart Hill Climbing – Restart the algorithm multiple times with different random initial states to increase the chance of finding the global optimum.

2. Simulated Annealing – Occasionally accept worse solutions to escape local maxima.

3. Stochastic Hill Climbing – Randomly choose among better neighbors instead of always selecting the best one, allowing for more diverse exploration.