Experiment

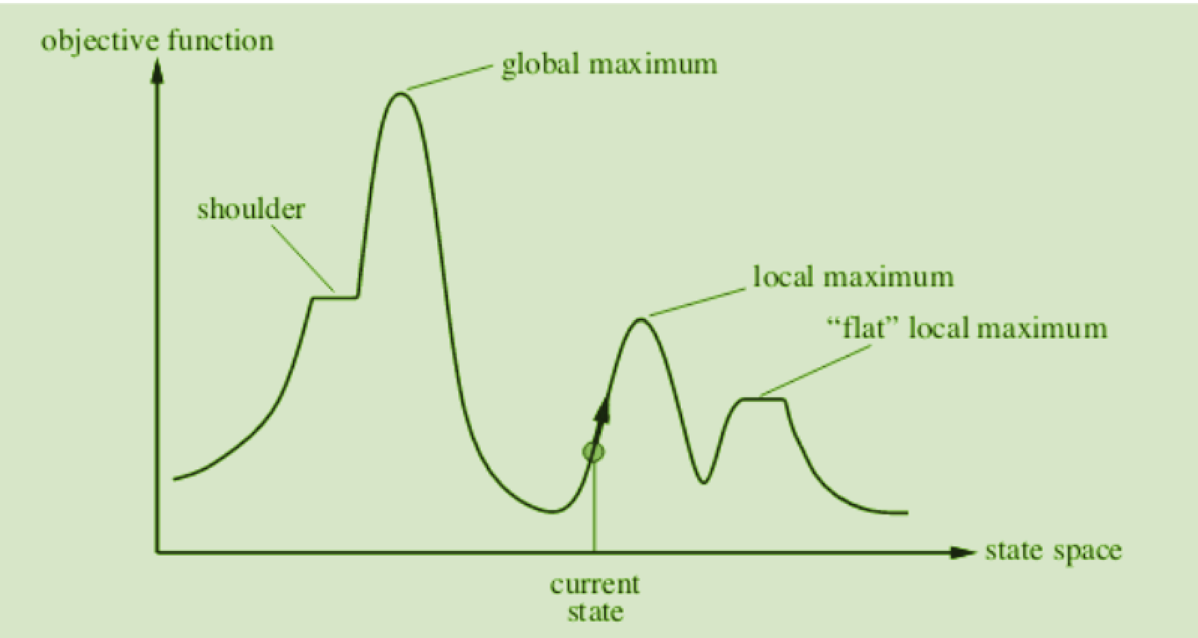
Aim: Implement a Local Search Technique (Hill Climbing)

Theory:

Local search techniques are optimization methods that operate by exploring neighboring solutions to iteratively improve a current solution. Unlike exhaustive searches, local search techniques are efficient for solving large problems by focusing only on local changes rather than searching the entire problem space.

Hill Climbing is one such local search technique where the algorithm starts with an arbitrary solution and iteratively improves by selecting neighboring solutions with a better objective value. The process continues until no better neighbors are found, indicating that a local optimum has been reached.

- **Goal**: Maximize or minimize a given objective function.
- **Challenge**: Hill climbing may get stuck in **local maxima** or **plateaus**, which



are not necessarily the global optimum.

Algorithm:

Initialization: Start with an initial random solution.

Evaluation: Evaluate the objective function for the current solution.

Generate Neighbors: Generate all neighboring solutions by making small modifications to the current solution.

Select Best Neighbor: Choose the neighbor with the highest improvement.

Termination: Repeat the process until no better neighbors exist (local optimum is reached).

Code:

```
import random


def hill_climbing(objective_function, solution, max_iterations=100, tolerance=1e-6,
restarts=0):

    current_solution = solution

    current_value = objective_function(current_solution)


    for _ in range(max_iterations):

        neighbors = generate_neighbors(current_solution)

        best_neighbor = current_solution

        best_value = current_value


        # Explore neighbors
```

```
        for neighbor in neighbors:
            neighbor_value = objective_function(neighbor)
            if neighbor_value > best_value:
                best_value = neighbor_value
                best_neighbor = neighbor

        # If no improvement, stop
        if abs(best_value - current_value) < tolerance:
            break

        current_solution = best_neighbor
        current_value = best_value

    # Optionally, perform random restarts to avoid local optima
    if restarts > 0:
        return random_restart(objective_function, max_iterations, tolerance, restarts,
current_solution, current_value)

    return current_solution, current_value


def generate_neighbors(solution, step_size=1):
    """Generates neighbors by adding or subtracting step_size from the solution"""
```

```
    return [solution + step_size, solution - step_size]


def objective_function(x):
    """Example objective function: A quadratic function with a maximum"""
    return -x**2 + 10*x + 5


def random_restart(objective_function, max_iterations, tolerance, restarts,
best_solution, best_value):
    for _ in range(restarts):
        initial_solution = random.randint(-10, 10)
        new_solution, new_value = hill_climbing(objective_function, initial_solution,
max_iterations, tolerance)
        if new_value > best_value:
            best_solution, best_value = new_solution, new_value
    return best_solution, best_value


# Run the hill climbing algorithm
initial_solution = random.randint(-10, 10)
result, value = hill_climbing(objective_function, initial_solution, max_iterations=100,
tolerance=1e-6, restarts=5)
print(f"Optimal solution: {result}, Objective value: {value}")
```

Output:

```
============ RESTART: C:/Users/Vanshita Singh/Desktop/hringkesh/7.py ===========
Optimal solution: 5, Objective value: 30
```

Conclusion:

In this experiment, the Hill Climbing algorithm was successfully implemented as a local search technique.

LO3, LO6 mapped.