Name: Jigar Siddhpura **SAPID:** 60004200155

DIV: C/C2 **Branch:** Computer Engineering

AI EXPERIMENT 4 - Hill Climbing

60004200155
Jigar Siddhpura

AI - Experiment 4

Aim: To implement local search algorithm - Hill climbing.

Theory: 1. It is a local search algorithm used for mathematical optimization problem.

2. It starts with an arbitrary solution to a problem & makes small changes to the solution. while & moving to direction of increasing elevation.

3. Basic steps of hill climbing:
   a) Initialization
   b) Generate neighbors
   c) Evaluate neighbors
   d) Move to best neighbor
   e) Repeat step 2-4 until there are no better neighbor or stopping criteria is met.

Advantages: a) Very easy to understand & implement
b) Doesn't require significant memory resources, making it suitable for problems with limited memory.

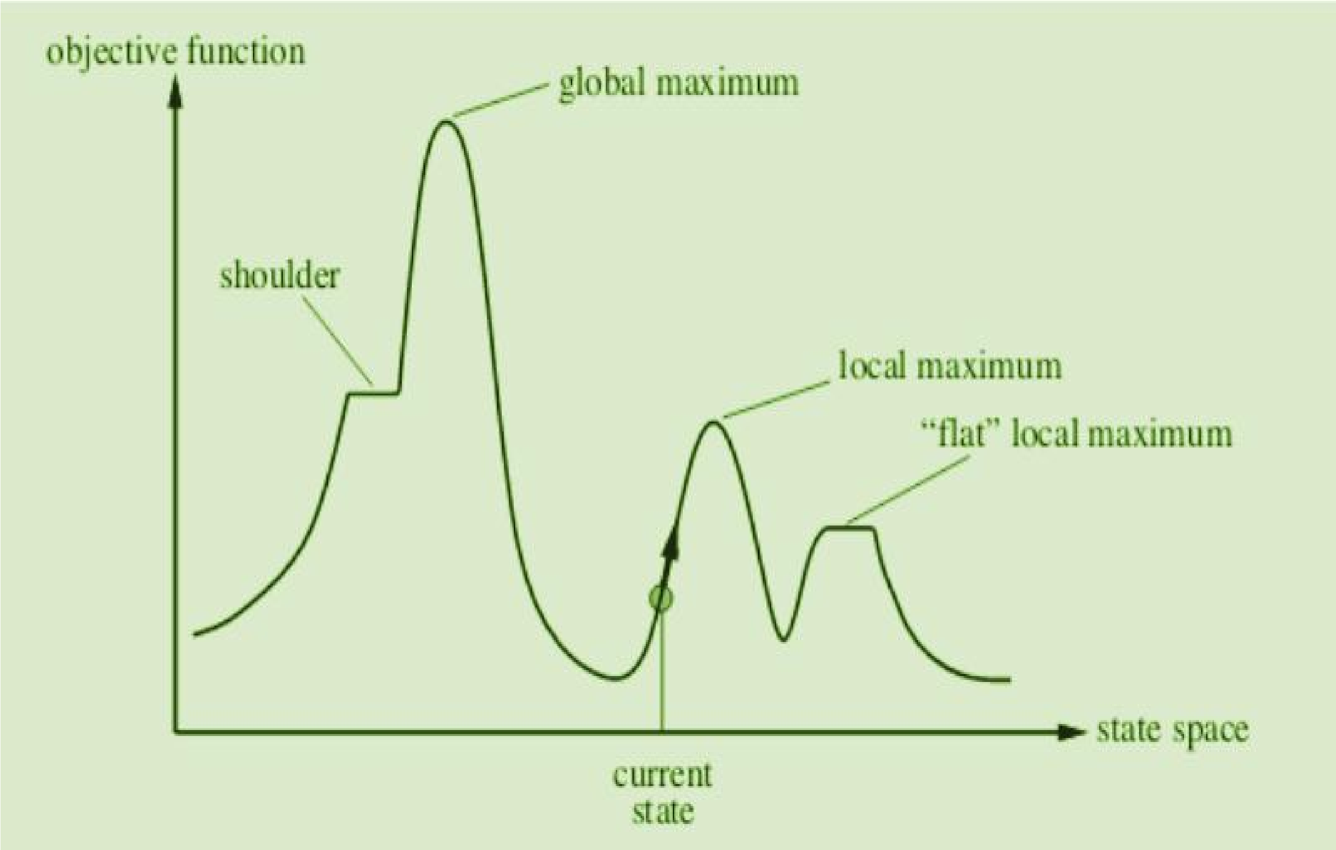
Limitations: a) It can get stuck in local optima & fail to find the global optima if search space is complex.

b) It has min. potential solutions because of no backtracking.

FOR EDUCATIONAL USE

Conclusion : It is serves as an excellent introductory algo & works well for specific types of problems. It is essential to be mindful of its limitations understanding the problems characteristics & exploring more sophisticated search techniques is crucial for tackling complex challenges, it proves as a solid foundation for learning local search algos.

FOR EDUCATIONAL USE



Code :

```
import copy

visited_states: list = []

def generate_child_state(current_state :list, prev_heuristic :int, goal_state :list) -> list|int :
    """Generates a child state by moving an element from one peg to another."""

    global visited_states
    state_copy = copy.deepcopy(current_state)

    for i in range(len(state_copy)):
        temp_state = copy.deepcopy(state_copy)

        if len(temp_state[i]) > 0:

            # .pop remove last element from the list and returns it *
            element = temp_state[i].pop()

            for j in range(len(temp_state)):
                new_state = copy.deepcopy(temp_state)

                if j != i:
                    new_state[j] = new_state[j] + [element]
                    current_heuristic = calculate_heuristic(
                        new_state, goal_state)

                    if current_heuristic > prev_heuristic:
                        child_state = copy.deepcopy(new_state)
                        return child_state
```

```python
    return 0

def calculate_heuristic(current_state :list, goal_state :list) -> int:
    """Calculates the heuristic value for the given state compared to the goal state."""

    goal_positions = goal_state[3]
    heuristic_value = 0

    for i in range(len(current_state)):
        check_values :list = current_state[i]

        if len(check_values) > 0:
            for j in range(len(check_values)):
                if check_values[j] != goal_positions[j]:
                    heuristic_value -= j
                else:
                    heuristic_value += j

    print(f"Heuristic value for {current_state} is {heuristic_value}")
    return heuristic_value

def solve_puzzle(initial_state, goal_state):
    global visited_states

    if initial_state == goal_state:
        print(f"Solution found! {goal_state}\n")
        Return

    # create deepcopy to prevent changes in the NESTED structure of the ORIGINAL list
    current_state = copy.deepcopy(initial_state)

    while True:
        visited_states.append(copy.deepcopy(current_state))
        print(f"Current State: {current_state}")


        prev_heuristic = calculate_heuristic(current_state, goal_state)
        child = generate_child_state(current_state, prev_heuristic, goal_state)

        if child == 0:
            print(
                f"No better heuristic value is obtained, declaring this as the goal state - {current_state}\n"
            )
            Return

        print(f"Child chosen for exploration: {child}\n")
        current_state = copy.deepcopy(child)


def main():
    global visited_states
    initial_state = [[], [], [], ['B', 'C', 'D', 'A']]
    goal_state = [[], [], [], ['A', 'B', 'C', 'D']]
    solve_puzzle(initial_state, goal_state)

if __name__ == "__main__":
    main()
```

Output :

```
PS D:\SEM 5\AI\EXPERIMENTS> python -u "d:\SEM 5\AI\EXPERIMENTS\hill_climbing.py"
Current State: [[], [], [], ['B', 'C', 'D', 'A']]
Heuristic value for [[], [], [], ['B', 'C', 'D', 'A']] is -6
Heuristic value for [['A'], [], [], ['B', 'C', 'D']] is -3
Child chosen for exploration: [['A'], [], [], ['B', 'C', 'D']]

Current State: [['A'], [], [], ['B', 'C', 'D']]
Heuristic value for [['A'], [], [], ['B', 'C', 'D']] is -3
Heuristic value for [[], ['A'], [], ['B', 'C', 'D']] is -3
Heuristic value for [[], [], ['A'], ['B', 'C', 'D']] is -3
Heuristic value for [[], [], [], ['B', 'C', 'D', 'A']] is -6
Heuristic value for [['A', 'D'], [], [], ['B', 'C']] is -2
Child chosen for exploration: [['A', 'D'], [], [], ['B', 'C']]

Current State: [['A', 'D'], [], [], ['B', 'C']]
Heuristic value for [['A', 'D'], [], [], ['B', 'C']] is -2
Heuristic value for [['A'], ['D'], [], ['B', 'C']] is -1
Child chosen for exploration: [['A'], ['D'], [], ['B', 'C']]

Current State: [['A'], ['D'], [], ['B', 'C']]
Heuristic value for [['A'], ['D'], [], ['B', 'C']] is -1
Heuristic value for [[], ['D', 'A'], [], ['B', 'C']] is -2
Heuristic value for [[], ['D'], ['A'], ['B', 'C']] is -1
Heuristic value for [[], ['D'], [], ['B', 'C', 'A']] is -3
Heuristic value for [['A', 'D'], [], [], ['B', 'C']] is -2
Heuristic value for [['A'], [], ['D'], ['B', 'C']] is -1
Heuristic value for [['A'], [], [], ['B', 'C', 'D']] is -3
Heuristic value for [['A', 'C'], ['D'], [], ['B']] is -1
Heuristic value for [['A'], ['D', 'C'], [], ['B']] is -1
Heuristic value for [['A'], ['D'], ['C'], ['B']] is 0
Child chosen for exploration: [['A'], ['D'], ['C'], ['B']]
```

```
Current State: [['A'], ['D'], ['C'], ['B']]
Heuristic value for [['A'], ['D'], ['C'], ['B']] is 0
Heuristic value for [[], ['D', 'A'], ['C'], ['B']] is -1
Heuristic value for [[], ['D'], ['C', 'A'], ['B']] is -1
Heuristic value for [[], ['D'], ['C'], ['B', 'A']] is -1
Heuristic value for [['A', 'D'], [], ['C'], ['B']] is -1
Heuristic value for [['A'], [], ['C', 'D'], ['B']] is -1
Heuristic value for [['A'], [], ['C'], ['B', 'D']] is -1
Heuristic value for [['A', 'C'], ['D'], [], ['B']] is -1
Heuristic value for [['A'], ['D', 'C'], [], ['B']] is -1
Heuristic value for [['A'], ['D'], [], ['B', 'C']] is -1
Heuristic value for [['A', 'B'], ['D'], ['C'], []] is 1
Child chosen for exploration: [['A', 'B'], ['D'], ['C'], []]

Current State: [['A', 'B'], ['D'], ['C'], []]
Heuristic value for [['A', 'B'], ['D'], ['C'], []] is 1
Heuristic value for [['A'], ['D', 'B'], ['C'], []] is 1
Heuristic value for [['A'], ['D'], ['C', 'B'], []] is 1
Heuristic value for [['A'], ['D'], ['C'], ['B']] is 0
Heuristic value for [['A', 'B', 'D'], [], ['C'], []] is -1
Heuristic value for [['A', 'B'], [], ['C', 'D'], []] is 0
Heuristic value for [['A', 'B'], [], ['C'], ['D']] is 1
Heuristic value for [['A', 'B', 'C'], ['D'], [], []] is 3
Child chosen for exploration: [['A', 'B', 'C'], ['D'], [], []]

Current State: [['A', 'B', 'C'], ['D'], [], []]
Heuristic value for [['A', 'B', 'C'], ['D'], [], []] is 3
Heuristic value for [['A', 'B'], ['D', 'C'], [], []] is 0
Heuristic value for [['A', 'B'], ['D'], ['C'], []] is 1
Heuristic value for [['A', 'B'], ['D'], [], ['C']] is 1
Heuristic value for [['A', 'B', 'C', 'D'], [], [], []] is 6
Child chosen for exploration: [['A', 'B', 'C', 'D'], [], [], []]

Current State: [['A', 'B', 'C', 'D'], [], [], []]
Heuristic value for [['A', 'B', 'C', 'D'], [], [], []] is 6
Heuristic value for [['A', 'B', 'C'], ['D'], [], []] is 3
Heuristic value for [['A', 'B', 'C'], [], ['D'], []] is 3
Heuristic value for [['A', 'B', 'C'], [], [], ['D']] is 3
No better heuristic value is obtained, declaring this as the goal state - [['A', 'B', 'C', 'D'], [], [], []]

PS D:\SEM 5\AI\EXPERIMENTS>
```

Conclusion :

Thus we successfully studied and implemented Hill-Climbing Search, and solved the block world problem with this algorithm.