Lab-4

Date 8/11/2024
Page

★ N Queens implementation using Hill climb algorithm

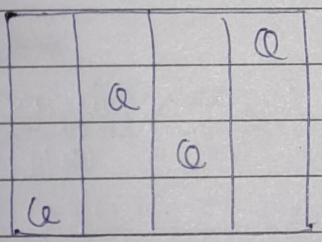
Algorithm for Hill climbing Algorithm

function Hill-climbing(problem) returns a state that is local maximum

current ← make.Node(Problem, Initial-state)
loop do
neighbour ← a highest valued successor
if neighbour.value ≤ current.value
then return state
end if
current ← neighbour
end loop.

Execute

Problem

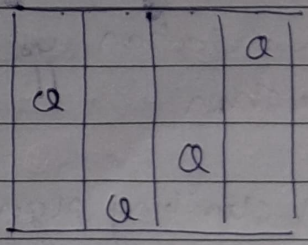


Initial state
$x_0 = 3, x_1 = 1, x_2 = 2, x_3 = 0$
cost = 2

neighbours

x_0	x_1	x_2	x_3	cost	
1	3	2	0	1	(choose)
2	1	3	0	1	
0	1	2	3	6	
3	0	1	0	6	
3	0	2	1	1	
3	1	0	2	1	

Date / /
Page

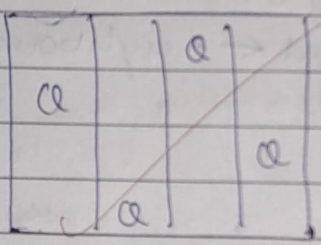


Chosen neighbour

neighbours

x_0	x_1	x_2	x_3	cost
3	1	2	0	2
2	3	1	0	2
0	3	2	1	4
1	2	3	0	4
1	0	2	3	2
1	3	0	2	0 // Choose

Solution



```python
print("Name: Sudarshan Komar", "USN: 1BM22CS291", sep="\n")

def count_conflicts(state):
    conflicts = 0
    n = len(state)
    for i in range(n):
        for j in range(i + 1, n):
            if state[i] == state[j]:  # Same column (vertical conflict)
                conflicts += 1
            if abs(state[i] - state[j]) == abs(i - j):  # Diagonal conflict
                conflicts += 1
    return conflicts

def generate_neighbors(state):
    neighbors = []
    n = len(state)
    for i in range(n):
        for j in range(i + 1, n):
            neighbor = state[:]
            neighbor[i], neighbor[j] = neighbor[j], neighbor[i]  # Swap queens i and j
            neighbors.append(neighbor)
    return neighbors

def hill_climbing(n, initial_state):
    state = initial_state
    while True:
        current_conflicts = count_conflicts(state)
        if current_conflicts == 0:
            return state
        neighbors = generate_neighbors(state)
        best_neighbor = None
        best_conflicts = float('inf')
        for neighbor in neighbors:
            conflicts = count_conflicts(neighbor)
            if conflicts < best_conflicts:
                best_conflicts = conflicts
                best_neighbor = neighbor
        if best_conflicts >= current_conflicts:
            return state, best_conflicts  # Return local minimum state and conflicts
        state = best_neighbor

def get_user_input(n):
    while True:
        try:
            user_input = input(f"Enter the column positions for the queens (space-separated integers between 0 and {n-1}): ")
```

```
            initial_state = list(map(int, user_input.split()))
            if len(initial_state) != n or any(x < 0 or x >= n for x in
initial_state):
                print(f"Invalid input. Please enter exactly {n}
integers between 0 and {n-1}.")
                continue
            return initial_state
        except ValueError:
            print(f"Invalid input. Please enter a list of {n}
integers.")

n = 4
initial_state = get_user_input(n)

solution = hill_climbing(n, initial_state)

if isinstance(solution, tuple):
    local_minimum_state, conflicts = solution
    print("No solution found (stuck in local minimum).")
    print(f"Local minimum state: {local_minimum_state}")
    for row in range(n):
        board = ['Q' if col == local_minimum_state[row] else '.' for
col in range(n)]
        print(' '.join(board))
else:
    print("Solution found!")
    for row in range(n):
        board = ['Q' if col == solution[row] else '.' for col in
range(n)]
        print(' '.join(board))
```

```
Name: Sudarshan Komar
USN: 1BM22CS291
Enter the column positions for the queens (space-separated integers
between 0 and 3): 0 2 0 3
No solution found (stuck in local minimum).
Local minimum state: [0, 3, 0, 2]
Q . . .
. . . Q
Q . . .
. . Q .
```