29-10-2024 WEEK- 6

Analyze and implement N-Queens problem using stimulated Annealing technique

ALGORITHM / PSEUDOCODE -

Date 29/10/24
Page 16

WEEK-5

Analyse and implement N Queens problem using stimulated Annealing technique.

ALGORITHM:

```
FUNCTION GenerateInitialState(n):
  Return a list of n random integers between 0 and n-1

FUNCTION Calculatecost (state):
  n = length of state
  cost = 0
for i from 0 to n-1:
  for j from i+1 to n-1:
    if state[i] == state[j] OR abs(state[i]-state[j]) ==
                                       abs(i-j):
        increment cost by 1
  return cost

FUNCTION Generate Neighbour (state):
   n = length of state
  new_state = copy of state
  row_to_stage = random integer between 0 and n-1
  new_state [row_to_change] = random integer
       between 0 and n-1
  return new_state

FUNCTION SimulatedAnnealing (n, initial_temperature,
                              cooling_rate):
  current_state = GenerateInitialstate (n)
  current_cost = CalculateCost (current_state)
  temperature = initial_temperature
```

```
IF neighbour_cost < current_cost:
    current_state = neighbour_state
    current_cost = neighbour_cost
ELSE:
    delta_cost = neighbour_cost - current_cost
    Probability = EXP(- delta_cost / temperature)
IF random number between 0 and 1 < probability:
        current_state = neighbour_state
        current_cost = neighbour_cost
temperature = temperature * cooling_rate
return current_state, current_cost

n = 8
initial_temperature = 100
cooling_rate = 0.95

Solution, cost = SimulatedAnnealing (n, initial_temperature,
                                     cooling_rate)
IF cost == 0:
    PRINT "Solution found:"
    PRINT solution
ELSE:
    PRINT "Could not find a perfect solution"
    PRINT "Best solution found:"
    PRINT solution
    PRINT "Cost:", cost
```

CODE-

```
import random
import math

def generate_initial_state(n):
  """Generates a random initial state for the N-Queens problem."""
  return [random.randint(0, n - 1) for _ in range(n)]

def calculate_cost(state):
  """Calculates the number of conflicts in the current state."""
  n = len(state)
  cost = 0
  for i in range(n):
    for j in range(i + 1, n):
      if state[i] == state[j] or abs(state[i] - state[j]) == abs(i - j):
        cost += 1
  return cost

def generate_neighbor(state):
  """Generates a neighboring state by randomly moving a queen."""
  n = len(state)
  new_state = state[:]
  row_to_change = random.randint(0, n - 1)
  new_state[row_to_change] = random.randint(0, n - 1)
  return new_state

def simulated_annealing(n, initial_temperature, cooling_rate):
  """Solves the N-Queens problem using simulated annealing."""
  current_state = generate_initial_state(n)
  current_cost = calculate_cost(current_state)
  temperature = initial_temperature

  while temperature > 1:
    neighbor_state = generate_neighbor(current_state)
    neighbor_cost = calculate_cost(neighbor_state)

    if neighbor_cost < current_cost:
      current_state = neighbor_state
      current_cost = neighbor_cost
    else:
```

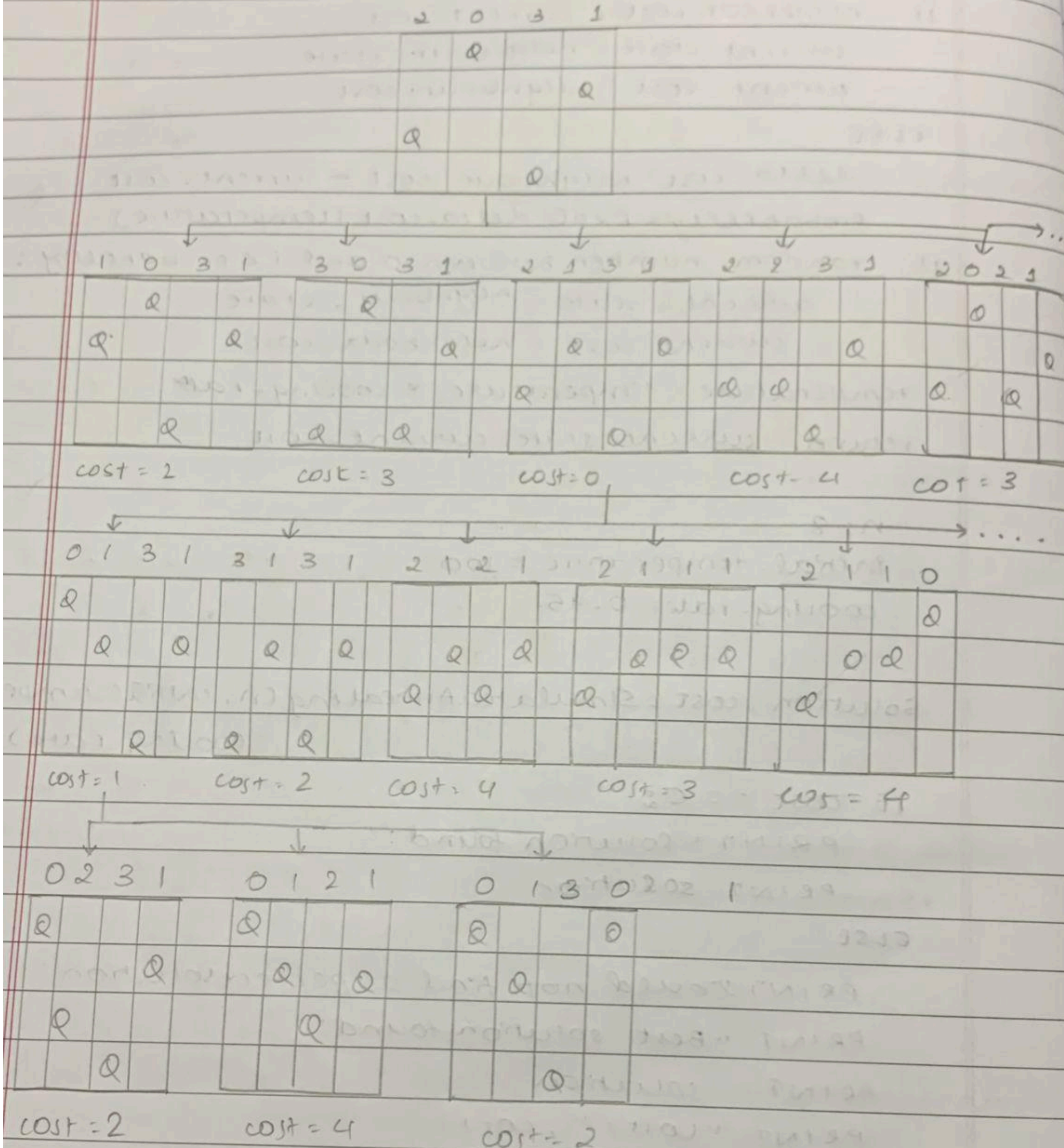
```
      delta_cost = neighbor_cost - current_cost
      probability = math.exp(-delta_cost / temperature)
      if random.random() < probability:
        current_state = neighbor_state
        current_cost = neighbor_cost

    temperature *= cooling_rate

  return current_state, current_cost

n = 8
initial_temperature = 100
cooling_rate = 0.95

solution, cost = simulated_annealing(n, initial_temperature, cooling_rate)

if cost == 0:
  print("Solution found:")
  print(solution)
else:
  print("Could not find a perfect solution.")
  print("Best solution found:")
  print(solution)
  print("Cost:", cost)
```

OUTPUT -

```
Could not find a perfect solution.
Best solution found:
[0, 7, 4, 2, 5, 6, 4, 0]
Cost: 3
```

STATE SPACE TREE -

STATE SPACE TREE:



Result = 2 1 3 1 cost = 0.