```
import random
import math

# Function to generate a random configuration of queens
def RandomConfiguration(N):
    # Each value in the list represents the row position of the queen in the respective column
    return [random.randint(0, N-1) for _ in range(N)]

# Function to calculate the number of conflicts (attacking pairs of queens)
def CalculateCost(state):
    conflicts = 0
    N = len(state)
    for i in range(N):
        for j in range(i + 1, N):
            # Check if queens i and j are attacking each other
            if state[i] == state[j] or abs(state[i] - state[j]) == abs(i - j):
                conflicts += 1
    return conflicts

# Function to generate a neighbor state by moving a queen to a new row in its column
def GenerateNeighbor(state):
    new_state = state[:]
    col = random.randint(0, len(state) - 1)  # Randomly select a column
    new_row = random.randint(0, len(state) - 1)  # Randomly select a new row for that queen
    new_state[col] = new_row
    return new_state

# Simulated Annealing function to solve the N-Queens problem
def SimulatedAnnealing(N, initial_temperature=1000, final_temperature=0.01, cooling_rate=0.995, max_iterations=10000):
    # Step 1: Initialize the board with a random configuration of queens
    state = RandomConfiguration(N)
    current_cost = CalculateCost(state)  # Number of conflicts (attacking pairs)

    # Step 2: Set parameters for the annealing process
    temperature = initial_temperature  # Start with a high temperature

    # Step 3: Annealing loop
    iteration = 0
    while temperature > final_temperature and current_cost > 0 and iteration < max_iterations:
        # Step 3a: Generate a neighbor state by moving one queen
        new_state = GenerateNeighbor(state)
        new_cost = CalculateCost(new_state)

        # Step 3b: Decide whether to accept the new state
        if new_cost < current_cost or random.random() < math.exp((current_cost - new_cost) / temperature):
            state = new_state
            current_cost = new_cost

        # Step 3c: Cool down (reduce temperature)
        temperature *= cooling_rate

        # Step 3d: Increment iteration count
        iteration += 1

    # Step 4: Return the final configuration and its cost
    return state, current_cost

# Example usage
N = 8  # 8-Queens problem
solution, cost = SimulatedAnnealing(N)
print("Final Solution:", solution)
print("Final Cost:", cost)
```

```
Final Solution: [1, 3, 5, 7, 2, 0, 6, 4]
Final Cost: 0
```

Start coding or generate with AI.