```
import random
import math

def initialize_state():
    """Initialize the state with a random configuration of the problem."""
    # Replace with specific initialization for your problem, e.g., N-Queens boar
    return [random.randint(0, n - 1) for _ in range(n)]

def cost_function(state):
    """Calculate the cost (or conflicts) of a given state."""
    # Replace with specific cost calculation, e.g., number of conflicts for N-Qu
    n = len(state)
    row_conflicts = [0] * n
    main_diag_conflicts = [0] * (2 * n - 1)
    anti_diag_conflicts = [0] * (2 * n - 1)

    conflicts = 0
    for row in range(n):
        col = state[row]
        row_conflicts[col] += 1
        main_diag_conflicts[row - col + n - 1] += 1
        anti_diag_conflicts[row + col] += 1

    for row in range(n):
        col = state[row]
        if row_conflicts[col] > 1:
            conflicts += row_conflicts[col] - 1
        if main_diag_conflicts[row - col + n - 1] > 1:
            conflicts += main_diag_conflicts[row - col + n - 1] - 1
        if anti_diag_conflicts[row + col] > 1:
            conflicts += anti_diag_conflicts[row + col] - 1

    return conflicts

def get_neighbor(state):
    """Get a random neighboring state by modifying the current state slightly."
    new_state = state[:]
    row = random.randint(0, len(state) - 1)
    new_col = random.randint(0, len(state) - 1)
    while new_col == new_state[row]:
        new_col = random.randint(0, len(state) - 1)
    new_state[row] = new_col
    return new_state

def simulated_annealing(initial_temp=1000, cooling_rate=0.99, max_iterations=10
    """Perform Simulated Annealing to find a solution."""
    current = initialize_state()
```

```
    T = initial_temp

    for i in range(max_iterations):
        if T <= 0:
            break

        next_state = get_neighbor(current)
        delta_E = cost_function(current) - cost_function(next_state)

        if delta_E > 0:
            current = next_state
        else:
            # Accept the worse state with a certain probability
            if random.random() < math.exp(delta_E / T):
                current = next_state

        # Decrease the temperature
        T *= cooling_rate

    return current
n = 4  # For 4-Queens problem
solution = simulated_annealing()
print("Final solution:", solution)
print("Number of conflicts:", cost_function(solution))
```

```
Final solution: [2, 0, 3, 1]
Number of conflicts: 0
```