EEC289Q-HW3

To address the Traveling Salesman Problem (TSP) for both graphs A and B, I would use a metaheuristic algorithm called Simulated Annealing (SA) combined with a greedy algorithm's initialization strategy. Simulated Annealing is a probabilistic technique for approximating the global optimum of a given function, particularly suitable for optimization and search problems. It simulates the physical process of heating a material and then slowly cooling it to minimize defects, thereby finding a global optimization of the solution. For graph A, considering the costs are Euclidean distances, geometric properties can be leveraged to optimize the search. For graph B, given that costs are randomly distributed, the algorithm needs good versatility and adaptability.

Algorithm Description

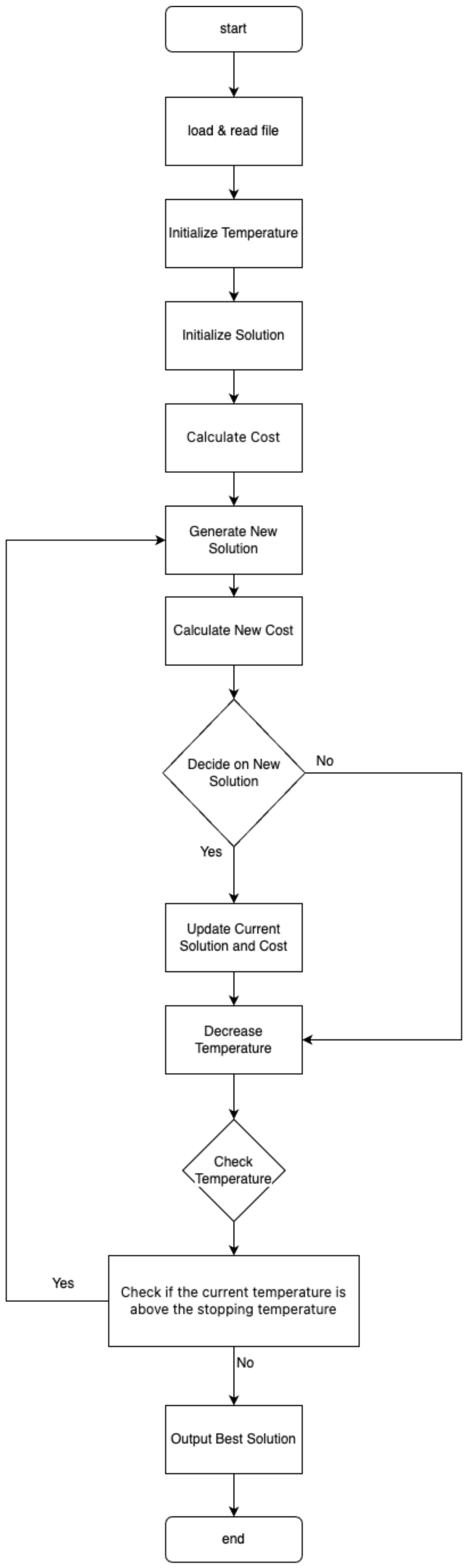
1. **Initialization**
    - Randomly select a starting solution as the current solution.
    - Set initial temperature and cooling rate.
2. **Iteration Process**
    - Randomly select a new solution within the neighborhood of the current solution.
    - Calculate the cost difference between the new and current solutions.
    - Accept the new solution as the current one if it is better.
    - Accept the new solution with a certain probability if it is worse; this probability decreases as the temperature lowers.
    - Decrease the temperature and repeat the above process until meeting the termination criteria (e.g., reaching the minimum temperature or time limit).
3. **Output Solution**
    - Output the best solution found during the iteration process as the final solution.

Algorithm Evaluation

- For graphs A and B, the number of cycles evaluated and the cost of the best cycle found will depend on the initial parameter settings, cooling speed, and the number of iterations. Within a 15-minute computation time, these parameters are adjusted to balance the breadth and depth of the search in hopes of finding the best possible solution within the given time.

Sample Flowchart and Pseudocode

1. **Flowchart**


start
load & read file
Initialize Temperature
Initialize Solution
Calculate Cost
Generate New Solution
Calculate New Cost
Decide on New Solution
No
Yes
Update Current Solution and Cost
Decrease Temperature
Check Temperature
Yes
Check if the current temperature is above the stopping temperature
No
Output Best Solution
end

Simulated Annealing (SA) Python code flow chart

1. **Start** - Begin the process.
2. **Load and Read File** - Load the file to create a distance matrix.
3. **Initialize Temperature** - Set the starting temperature for the simulated annealing process.
4. **Initialize Solution** - Generate an initial solution randomly.
5. **Calculate Cost** - Calculate the total distance (cost) of the current solution.
6. **Generate New Solution** - Swap two cities in the solution to create a new potential solution.
7. **Calculate New Cost** - Calculate the total distance (cost) of the new solution.
8. **Decide on New Solution** - Determine whether to accept the new solution based on its cost and the current temperature.
    - If the new cost is less than the current cost, or if a probability condition based on the temperature and cost difference is met, accept the new solution.
9. **Update Current Solution and Cost** - If the new solution is accepted, make it the current solution and update the current cost.
10. **Decrease Temperature** - Apply the cooling rate to decrease the temperature.
11. **Check Temperature** - Determine if the current temperature is above the stopping temperature.
    - If yes, go back to step 6 to generate a new solution.
    - If no, proceed to the next step.
12. **Output Best Solution** - Output the best solution found, its cost, and the number of iterations (cycles) evaluated.
13. **End** - End of the process.

2. **Pseudocode**

```
BEGIN
  Initialize current_solution, best_solution
  Set initial_temperature, cooling_rate, minimum_temperatu
re
  WHILE (temperature > minimum_temperature)
    Generate new_solution from the neighborhood of current
_solution
    Calculate cost_difference between new_solution and cur
rent_solution
    IF (new_solution is better OR exp(-cost_difference / t
emperature) > random number)
      current_solution = new_solution
      IF (current_solution is better than best_solution)
        best_solution = current_solution
    ENDIF
    Reduce temperature according to cooling_rate
  ENDWHILE
  OUTPUT best_solution
END
```

Result

▼ 1000_randomDistance.txt file

`Best cost: 39764.26`

`Iterations (cycles) evaluated: 9.17e+02`

▼ 1000_euclidianDistance.txt file

`Best cost: 44050.4`

`Iterations (cycles) evaluated: 9.17e+02`