# INFO 6205 Spring 2023 Project

# *Traveling Salesman*

Changyu Wu 002111594, Ruizhe Zeng 001241850

## I. Intrduction

### Aim

Travel salesman problem describes a situation that if given a set of cities, what is the shortest path to go through all the cities exactly once. It has been considered one of the most difficult problems for the computer to solve, since it is a NP-hard problem. If we try to solve it brutally by getting all the possibilities and comparing them, it will take an incredibly long time for even a very small number of cities. Luckily we can get a good enough solution instead of perfect in a reasonable time by following those steps in the next paragraph.

### Approach

1. Import a csv file to create a graph with cities as nodes and distances between them as edges.

2. Apply Prim algorithm to generate a Minimum Spanning Tree (MST).

3. Create a multi-graph from the MST that only contains nodes with even degree except for solo nodes. Apply a matching algorithm to the multi-graph to generate a perfect matching solution.

4. Generate the first version of solution by starting at any node and traversing the graph according to the matching pairs.

5. Apply tactical and strategic optimizations including random swapping, 2-opt, genetic algorithms and ant colony algorithms to improve the tour's distance.

## II. Program

### Data Structures & classes

Class Graph represents a graph data structure. It has the following fields: “nodes” is an ArrayList of Node objects that represents the vertices of the graph. “edgesMatrix” is a double array representing the edges of the graph. Each cell (i,j) contains the weight of the edge between node i and node j. “graph” is an array of LinkedList objects that represents the edges of the graph. Each linked list contains the edges that are adjacent to the corresponding node. There are several key methods in this class. “createGraph” creates a graph from a CSV file. It reads the file and populates the nodes field with Node objects. It then populates the edgesMatrix and graph fields by computing the distance between each pair of nodes.

The Node class represents a node in a graph. It has four instance variables: a unique ID, longitude, latitude, and an ID. ID and unique\_id are both used to identify each node in the graph. The difference is unique\_id is used to provide a human-readable name or identifier for the node. The longitude and latitude are used to specify the location of the node. The class has a constructor that takes the ID, longitude, latitude, and unique ID as parameters and initializes the instance variables. It also has a toString() method that returns a string representation of the node object. There are also four getter methods: getId(), getLongitude(), getLatitude(), and getUnique\_id(), that return the values of the corresponding instance variables. These methods allow the user of the class to access the node data in a safe and controlled manner.

### Algorithm

Prim:

In computer science, Prim's algorithm (also known as Jarník's algorithm) is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.[1]

In our implementation, the graph is represented as an adjacency list of double arrays, where each array contains information about an edge: the index of the node it starts from, the index of the node it ends at, and its weight. The algorithm maintains a priority queue of cut edges, which are edges that connect a visited node to an unvisited node. The queue is maintained by the weight of the edges, and the algorithm adds the minimum-weight edge to the tree at each step. The algorithm uses a boolean array to keep track of visited nodes and a list of linked lists to represent the minimum spanning tree. The method cut() adds all the cut edges of a node to the priority queue.

The time complexity of Prim's algorithm is O(E logV), where E is the number of edges and V is the number of nodes in the graph, and the space complexity is O(E+V). This implementation uses a priority queue to store the cut edges, which has a worst-case time complexity of O(logE), and a boolean array to store visited nodes, which has a space complexity of O(V).

GreedyMinWeightMatching：

Takes the MST created in Prim and stores it in its private variable graph. Call isolateOddDegreeNode(), it will return all nodes if its number of edges cannot divide by 2. Then call findMinWeightMatching() to find the minimum weight matching for odd degree nodes. Notice that there are more than one algorithm we can pick for this, Blossom algorithm or Greedy. We choose Greedy because Blossom is harder to implement and it takes O(n^3), instead Greedy is simpler and it takes only O(n^2). Even if we use Blossom algorithm there is no guarantee we can get the shortest path, and we have more improvement methods later so we can save us some time in this step. Finally mergeEdges() will simply merge two graphs by adding the edges. At the end it forms a multi-graph.

Random Swapping:

Random Swapping is a local search algorithm that attempts to improve the initial TSP solution. The algorithm repeatedly randomly swaps two cities in the current best tour and evaluates the new resulting tours. It terminates when no improvements can be found for a certain number of iterations, as determined by the cnt variable. The algorithm's effectiveness depends on the quality of the initial solution and the specific problem instance.

2-Opt:

2-Opt is another local search algorithm that tries to improve the initial TSP solution.The main idea behind it is to take a route that crosses over itself and reorder it so that it does not. A complete 2-opt local search will compare every possible valid combination of the swapping mechanism.[2] It works by iteratively swapping pairs of edges in a tour until no further improvements can be made. The algorithm performs a 2-opt swap when the length of the two edges AB and CD plus is greater than the length of the two edges AC and BD. If a swap is made, the algorithm updates the current best tour and distance. The process is repeated until no further improvements can be made.

Genetic:

In Tactical OPT we will record a list of good solutions, they must have a shorter path compared with the solution obtained in Tour. We put those solutions in the parents variable in Genetic. In getCrossover(), for every pair of parents we do a crossover which picks a random part of parent a, removes all nodes of changed part in parent b and adds changed part at the tail of parent b. In getMutation(), for every parent we do a mutation which takes a random part of parent a, and puts it back in a random index. For every run, we also call getNewParent() which shuffles existing solutions to get a new random solution. We put all childs from mutation/crossover/getNewParent and put them into a priority queue, and for each run we only keep the top 100 best solutions and put it into the population. We repeat steps until the goal number of runs is reached.

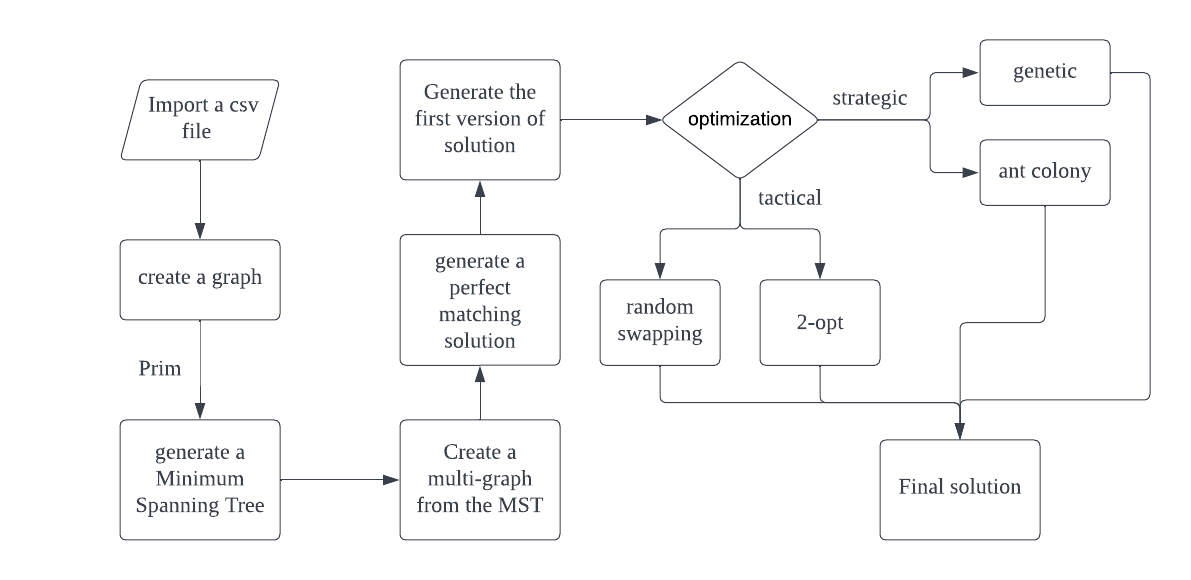
Ant Colony:

We will have a phMatrix[n][n], n = num of nodes, initially all 1.We put an ant in every node, each ant will complete a tour. Every time an ant will randomly pick a node by following rules: 1. tij = phMatrix[i][j] 2. assume there are n possible picks the chance it picks the first node is tij\_1\*(1/distance\_1)/tij\_1\*(1/distance\_1)+tij\_2\*(1/distance\_2)+...+tij\_n\*(1/distance\_n). Then we do a cumulative Sum so the result will be looking like {1,0.9,0.56…..}.The first element is always equal to 1. Then we generate a random double in range 0 to 1. if 0.9<random double<1 picked first ,if 0.9<random double <0.56 pick second…etc.After all ants finished their tour, we update phMatrix by (1-p)\*tij + total(1/distance). total(1/distance) is sum of 1/solution get by a ant’s distance. We also repeat all steps until the goal number of runs is reached.

### Invariants

Nodes and original graph(all possible edges).

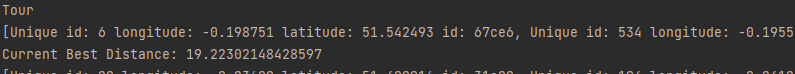
## III. Flow Charts



Graph 1. Flow Chart

## IV. Observations & Graphical Analysis

Base on our observation the tour we get from Christofides’s algorithm has lots of place to improve, for ex:



Graph 2. Christofides’s algorithm result



Graph 3. Christofides’s algorithm result

The solution we get form Christofides’s algorithm is around 19.223, but we can find a solution that’s only 7.63 later.

The four improvement method we choose is :

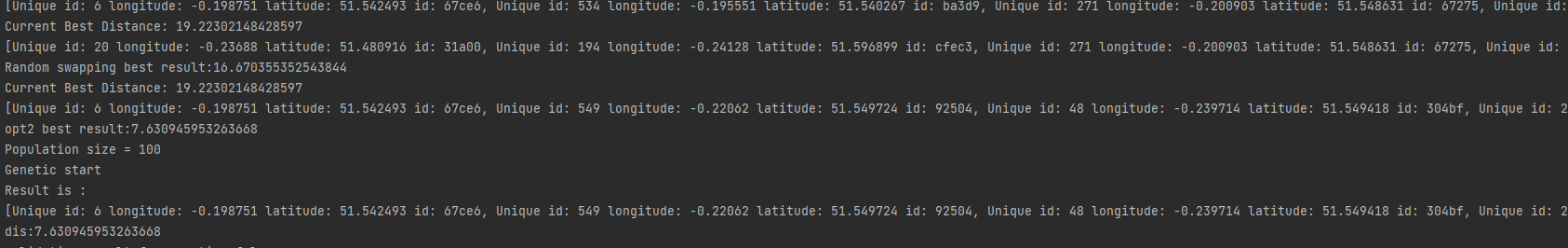
* Random Swapping
* 2-opt
* Genetic algorithm
* Ant Colony algorithm

Based on our experiment, even theoretically Random Swapping is able to find a very short solution if we are super lucky, but overall 2-opt has much better behavior. And the result from 2-opt is very close to the best solution we can find. In the set of 585 data, we cannot find any better solution than 2-opt even if we run 1000 times of Genetic/Ant Colony algorithm. In the set of 156 data, we can find a better solution using the genetic algorithm, which decreases the best solution from 4.63 to around 4.3.

Ant Colony algorithm has potential to find a good solution but it is too slow. At the first we all Tijs to 1. Tij is the pheromone's value stored in the phMatrix. Every time we find a valid path we increase the Tij to Tij +1/distance of path. However it is too slow, because the start value is huge (around 60). Then, we decide to get top 100 solutions from 2-opt and use them to update phMatrix before starting ant colony, which decreases the start value to around 40, but still the distance decreases very slowly between loops. So we decide to put in more pheromone every time we find a path. When updating the matrix we let Tij = Tij+25/distance, but still very slow. So our best prediction to get a best the solution will be:

Christosfides’s algorithm -> 2-opt -> Genetic algorithm.

## V. Results & Mathematical Analysis



Graph 4. Christofides’s algorithm result



Graph 5. Christofides’s algorithm result



Graph 6. Christofides’s algorithm result



Graph 7. Christofides’s algorithm result

As you can see the best result we can find is 7.63, at set 585 even if we run genetic and ant colony 1000 times , we cannot find a better result.

**Time of Christofides’s algorithm cost (n = number of nodes)**

Create total graph:O(n^2)+ Prim:O(n^2) +Greedy min weight matching O(n) + generate tour:O(n) = O(2n^2+2n)

**Time of random swaping cost :** O(count) ;count = input count value

**Time of 2-opt:** O(n\*(n-1))

**Time of Genetic algorithm:**

CrossOver: O(n\*(n-1)) + Mutation: O(n) + New Parent O(1) = approximately O(n^2) per loop

**Time of ant colony:**

n ants \* n moves per ant + update phMatrix:n^2 = approximately O(n^2) per loop

## VI. Unit tests

The test method is written in JUnit called testTSP. The first part of the test method creates a graph by reading data from a CSV file and then applies the Prim algorithm to create a minimum spanning tree. It then builds a multi-graph by including only even degree nodes except for a solo node.

Next, it generates a tour using the multi-graph, and applies different optimization techniques to the tour such as random swapping, 2-opt, genetic algorithm, and ant colony optimization algorithm. For each optimization technique, it prints the best result (shortest distance) and the resulting path. Finally, it uses the assertTrue method from the JUnit framework to assert that the resulting distances are correct and that the paths have valid solutions.

The purpose of this unit test is to verify that the TSP solution works correctly by testing various optimization algorithms on the TSP problem and comparing their results against each other.

## VII. Conclusion

In conclusion, the Traveling Salesman problem is a difficult problem for computers to solve since it is an NP-hard problem. In this project, we have implemented a solution that generates a good enough solution in a reasonable amount of time. The solution involves importing a csv file to create a graph with cities as nodes and distances between them as edges. We then applied the Prim algorithm to generate a minimum spanning tree (MST), created a multi-graph from the MST that only contains nodes with even degree except for solo nodes, applied a matching algorithm to the multi-graph to generate a perfect matching solution, and generated the first version of the solution by starting at any node and traversing the graph according to the matching pairs. Finally, we applied tactical and strategic optimizations including random swapping, 2-opt, genetic algorithms, and ant colony algorithms to improve the tour's distance. Through our implementation, we have demonstrated the use of different algorithms and data structures, as well as the effectiveness of combining different techniques to improve the solution. By comparison, the 2-opt and genetic optimizations are fast and accurate. Ant colony optimization is too slow, but still could reach good results.

## VIII. References

[1] Prim, R. C. (1957). Shortest connection networks and some generalizations. The Bell System Technical Journal, 36(6), 1389-1401.

[2] Johnson, D. S., & McGeoch, L. A. (1997). The Traveling Salesman Problem: A Case Study in Local Optimization. In Local Search in Combinatorial Optimization (pp. 215-310). John Wiley & Sons, Ltd.