# INFO 6205 Spring 2023 Project

# *Traveling Salesman*

Changyu Wu 002111594, Ruizhe Zeng 001241850

**Github link**:https://github.com/Lethe1999/INFO6205-Final-Project.git

## 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.

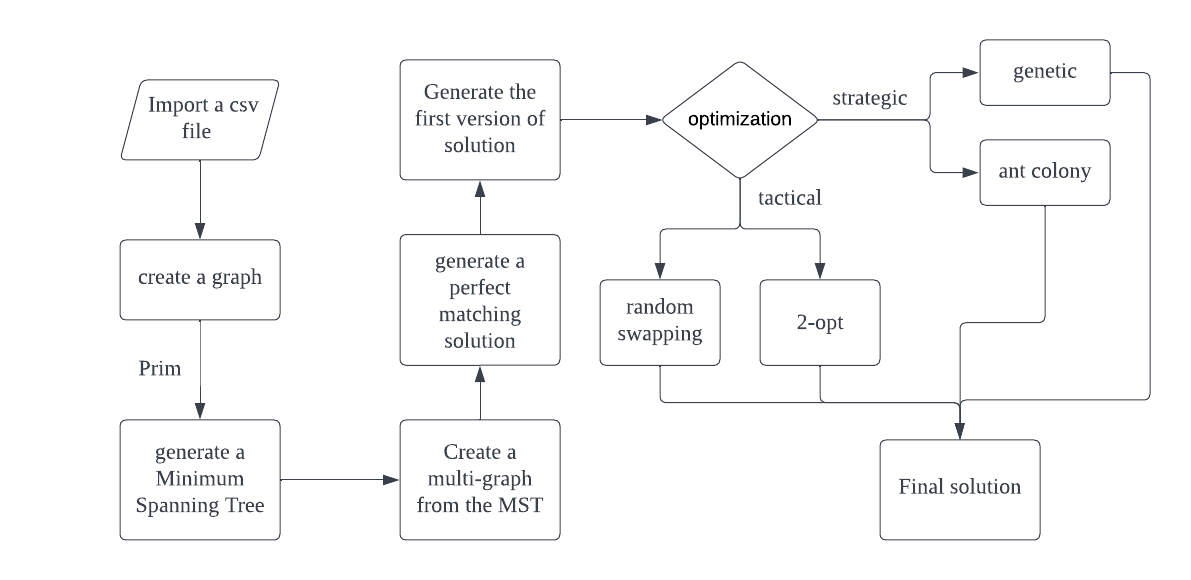
Ant Colony:

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).

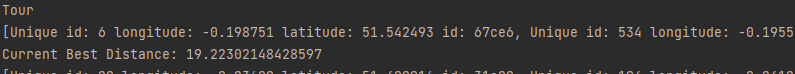
## III. Flow Charts



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 :

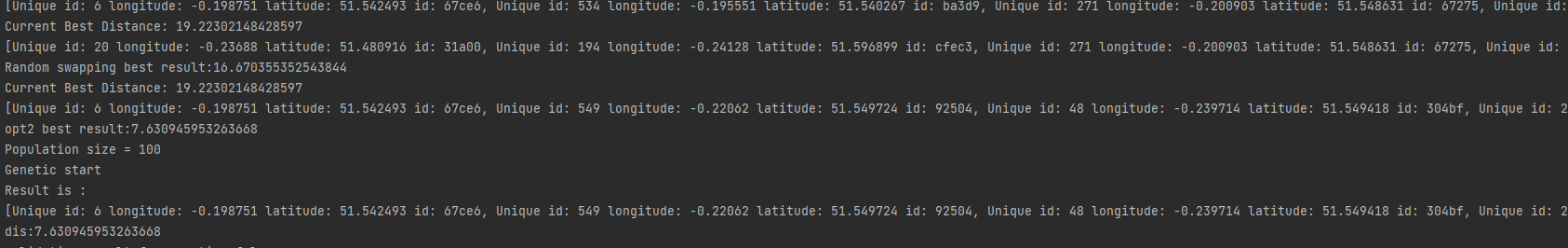
* Random Swapping
* 2-opt
* Genetic algorithm
* Ant Colony algorithm

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 10000 times of Genetic/and 1000 times of 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

**Best Path we found:**

67ce6

92504

304bf

3a16f

f47a9

c3faf

cfec3

861e4

cce32

ba74e

ba542

30704

2b603

b82a2

6b397

74a15

1512a

ceab2

d9536

e71b5

36c1d

c9b4d

ba8bd

67275

ba3d9

bf0de

7e786

1ff43

95134

67e12

116db

ca409

9302e

e0a06

f4d9e

5291f

a4cdd

1f4d0

3b420

44dd1

7c283

7be72

ae31c

fd53c

7dfe2

0c987

2bf09

69f18

394fd

99d29

486aa

61a04

90125

a38d8

ccae0

15c85

4dd65

e924d

aa7ba

4b986

d24e7

7d2d7

3f4fb

3f884

5f5cc

3f894

7ac7b

5f5b4

95ba0

1a5df

e269e

4fa0b

2cc86

d102d

03c99

0868c

4a714

c0b73

79045

f8184

a47d7

33902

1f724

0637b

d972b

436ec

97f15

0f199

4afb3

b8962

d0993

af279

9fe9a

a7ebe

e1cd5

5ffeb

0db30

87975

90290

56bcd

3ffb2

61f08

8e6f9

5e408

9b30d

bebb5

96939

0ad97

a8c25

e280f

23239

b4b46

86e93

bbb28

e53f8

12325

15e84

d0c60

292c6

27e0c

0ee10

36116

5099b

1c795

15f40

ab447

1d259

fe864

fdde0

4709f

bad43

bf103

55f0d

f00de

cd848

4ee9a

54e6b

f4456

99bdc

ec311

4521a

c59bd

5da55

6389b

7a64e

b50ca

e6000

19884

e71b5

b61cf

7713f

3849e

814da

5fe9b

41072

6616d

48feb

7b357

4cf86

bc3a7

f5094

26d4a

09a87

31a00

1a8db

f5d8a

ceae0

7a813

b465f

824ba

bd449

3ff9d

f76fa

04f3f

0598c

ba3e8

a517b

c4459

95f4d

e2d42

ff988

e5200

97951

395c9

22e48

22e22

c232d

dc716

c2856

9410e

50735

29946

810b6

8e3a4

2cc42

7fc2d

49fc4

73b8a

12ddb

10c55

7fe42

5db1a

af3c2

846ba

8a2d7

4ab5a

46b61

57060

10f92

7a2b1

19308

c7f7f

458d6

67403

5b793

3ce3f

bb937

fb56c

677a4

9182f

6b010

7d25b

8c1e3

79146

b560c

30d1a

480e3

6e8c2

05905

80a9e

08086

8df36

a5dd1

f02c2

60d52

35be3

02fe1

4e77b

573ab

b7969

eae5e

71547

604fc

2972a

9e57f

144e9

d70ae

a7f6f

3e026

0e16a

5042a

31fc0

98adb

f068b

0c7b9

7cb4d

cad0c

9be80

6e85a

0b647

fd1d4

cbe21

ead04

b559f

faea3

c0da2

fdf68

5b318

da0ba

c7b1e

8f6cc

8d330

bb562

95d5b

78be7

3ba9e

7367c

a7bac

6742c

63f62

e70a8

527ca

03f24

3db46

e228c

15ff6

f6f39

73c88

d58f6

957cf

bc97f

f4a14

a14f0

70700

c543e

f384c

ea746

916a4

a02b4

47823

64e7a

f0ed5

59add

22d14

15ede

71eab

0428d

01c6c

aae64

bb3a6

0a7a9

8e105

f98e7

faeaa

7283d

b71c9

6c267

53135

b85d9

b7681

ba063

7040f

af4ce

0951a

02057

9db58

29943

ae1e1

b5bfd

8b9eb

7fa41

a0647

d3daf

09e85

afcc4

647af

84393

b51e0

b50c4

0ad98

1b369

57aa7

bff1e

198db

ffb25

f1a4e

97aa5

b1bf9

f4751

46e26

0d210

e8c5b

eac9b

adc1f

3dd82

63a49

93619

ff1b8

ccf12

b1b9b

e8d70

c8783

ea5e7

da011

780a4

6ed74

8fb58

874e3

681a5

09250

10777

98ae6

e9a50

a75a5

c3355

cbe84

1eb20

8c81e

3f416

54520

a6608

d9ff6

454af

372a3

68ff5

57bc7

6dc46

9c5a3

ac2e5

36aca

7fcd4

29847

3361b

96dd4

bd42e

88b5c

789d9

4dddd

c80c3

6cb16

1e779

c726b

58d22

62a5f

66ceb

7cd8b

f2fb7

a661b

9e912

1fc21

dbd67

d4299

42ba6

3a074

ed4c1

f1507

cfcbe

3d168

803cc

595f8

cec4c

c15c0

2ccf2

cdfcf

c5f90

e701b

2d349

421bf

09e90

18474

ffe9e

d8c20

e1b9b

af637

61e92

662e9

fe7ae

8d2d7

50ed1

801c9

d1ba6

d9298

43aa8

0c956

3a496

76697

14b0b

d2c1d

9bdee

a0def

f3ee1

4d2c0

ca598

dded9

ef0b1

8bdfb

3eee1

bdafb

83178

f5b0f

fdcf4

5da87

ba3ae

53257

a833d

2d0ae

2a25c

34272

e8c89

b06c7

f68b8

c5c16

2cf9b

b97cd

400a2

d4615

7773f

b46af

1c7e4

4dffe

19cb3

5439a

d8bd1

9e51c

4dcaf

e85ee

44fb2

36cae

cd05f

fa4e7

e61cc

a7b0f

db60f

60f32

2db1d

f2703

594af

85357

f7d41

a7ca1

551e0

9654a

f546d

703b4

d9396

4bcd6

42640

6a7d5

cc9f3

4fc27

38c6e

e0ca5

049d1

29645

a30f9

f9816

ec3b1

c3298

6afd9

5aad7

63b73

110be

a1727

6b450

3927f

d2f04

d816e

93956

937c0

b37e5

d6ebf

31838

5bfe3

cfdb9

afe1b

e5239

0bcd0

4bdb7

b5df9

fb049

d7b8b

b8ffc

328a2

d5aeb

f22f2

6c3a7

4972e

829fb

fcbbe

9686e

1eaf0

497b9

b4447

d7bfd

44234

70cf0

790fd

6d4ba

35c5c

ceaeb

8064f

e93d1

c51e6

d1eaa

987b5

f37cc

c828a

cee38

74aa7

b9086

ca21a

2e6f4

d15fd

9a790

c80d0

35c3d

8d4c3

ff749

51f69

9526d

4b883

60f63

3a633

f0621

63cb4

## 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.