Srinivasan Sundar
1217212009

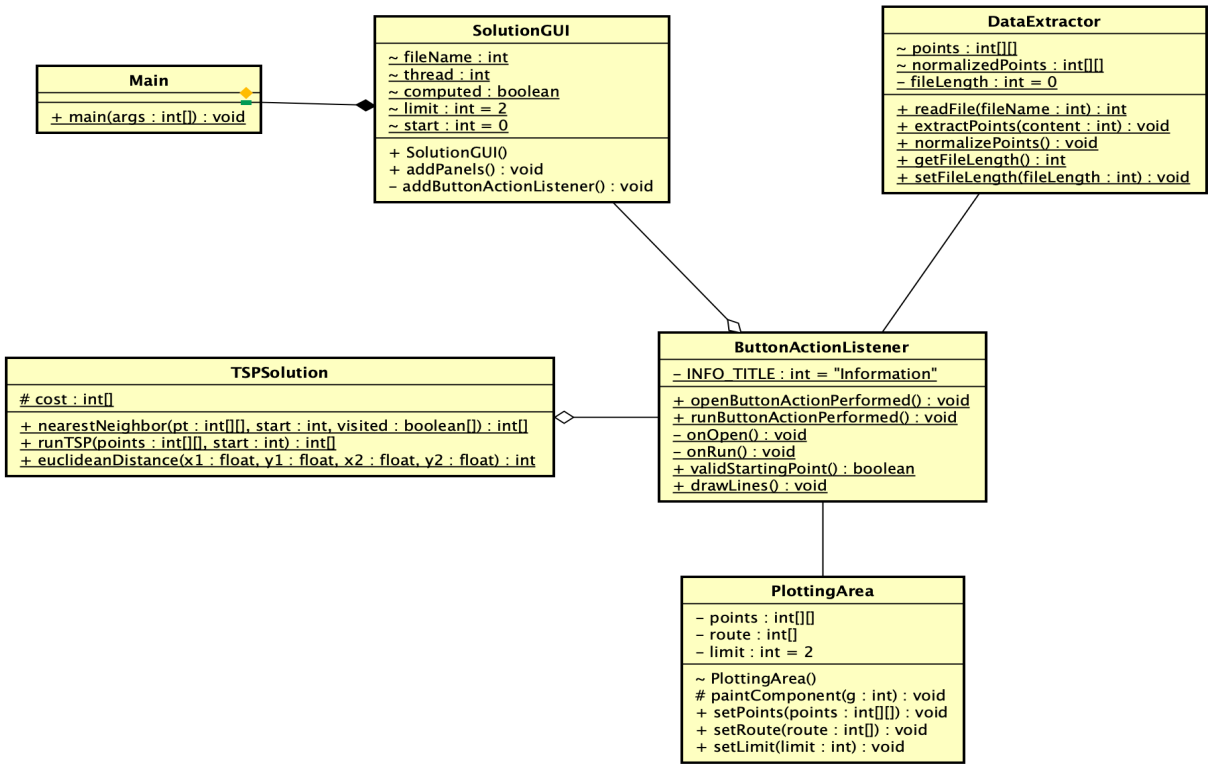
Assignment 2

User Stories:

As a user, I want to open a symmetric file, so that I can select the file for which I want output.
As a user, I want to view the points present in the opened file.
As a user, I want to select the starting city for the travelling salesman problem.
As a user, I want to run the nearest neighbor algorithm on the opened file.
As a user, I want to view a step-by-step animation of how path was created.
As a user, I want to view the number of iteration and best distance in each step of the animation.
As a user, I should be able to pause the animation while nearest neighbor algorithm is running.
As a user, I should be able to resume the animation after pausing the animation.

These user stories are created based on the requirements discussed in the class. All these user stories follow INVEST principle. Each user story is small and independent of each other. User stories are value to the user. We can estimate how much time each user story would take to complete and also, we can test whether we have fulfilled the requirement of the user story.

Class Diagram:



Class Explanation:

There are 6 classes in this solution. Each class is connected to each other through Association, Aggregation, Composition. Main and SolutionGUI class is connected with aggregation. SolutionGUI and TSPSolution is connected with Composition connection. ButtonActionListener and TSPSolution is connected with Composition connection. ButtonActionListener is connection to PlottingArea and DataExtractor using Association. Each class has very low interdependency with each other.

Low Coupling:

Low coupling is achieved in my solution by having low interdependency between each class. Each class is designed based on the specific functionality it the individual responsibility. Each class is independent of each other. The interaction between each class is through the objects passed through the method parameters. By this way the class will have low dependency when compared to creating the class instance each time. Each class are designed in such a way that it has a single responsibility to perform.

Explanation about coding:

Clean Coding:

I implemented clean code by using KISS (Keep It Simple) and DRY (Don't Repeat Yourself) principle. All the functionality that is implemented clean, clear and simple to understand. Whenever a long code was found I decomposed into different function based on the functionality. I had made sure that the code is simple to understand, and it can be modified by other developers easily. My solution does not have any repetition of the code. The solution was scanned with SonarLint Plugin to scan for any improvements that can be enforced in the implementation. I implemented sonarLint suggestions that are necessary and are required to be implemented.

Stepwise Refinement:

DataExtractor module is decomposed into smaller functions based on the functionality each function should perform. DataExtractor has 5 functions in total. For reading file content we have a function 'readFile', which opens a file and returns the content of the file. For Symmetric data, 'extractPoints' function extracts the points of each city and forms an integer array. NormalizePoints function normalizes the extracted points based on the window size, so that all points can be plotted in the window. TSPSolution module is decomposed into 3 functions. The main function interacts with the user to get the file that needs to be processed. NearestNeighbor function computes the shortest distance to connect all the cities and provides the route. ButtonActionListener is decomposed to 5 functions for performing various function. Each function interacts with each other by function call and passing the parameters into the function. Each function is decomposed to smaller units based on the functionality.

Modularization:

Solution contains six modules. Modules are divided based on the responsibility it should handle. DataExtractor module is responsible for handling the data extraction for 'tsp' type of data files. This module reads the contents of file and process it contents based on the file type. This responsibility of the DataExtractor is independent of the TSPSolution module. Any changes that is related to algorithm of connecting the cities are taken care by the TSPSolution module. TSPSolution module interacts with DataExtractor module by accessing the static points variable. The SolutionGUI and Main module in this solution is the GUI that interacts with the user and used for visualization of the solution. ButtonActionListener has functionality to open the file and use that data for calculating the TSP solution. It also updates the state in the view by updating the drawing panel based on the data points in the solution. Since all modules are independent of each other and the information handled by one module is not reflected on another module. By this way, Information hiding is achieved in this solution.

Top-Down Approach:

I used top-down approach to implement this solution. In top-down approach the general requirement is refined and decomposed to specific task that needs to be implement. These small tasks are implemented in order to accomplish the overall requirement of the solution. For instance, opening of file user story is decomposed to adding button to the GUI, opening file using JFileChooser, storing the filename in a variable which will be used by other components of the solution. Likewise, I have broken down the requirements into smaller task and worked on the smaller tasks to complete the requirement. The solution I have implemented can be broken down into the following:

- User interaction like open, run, stop, starting point input through GUI,
- Data extraction from the input file,
- Applying the nearest neighbor algorithm on the input dataset
- Plotting points and route based on the state of the data.
- Update the drawing panel with the route lines for each second using the thread concept.

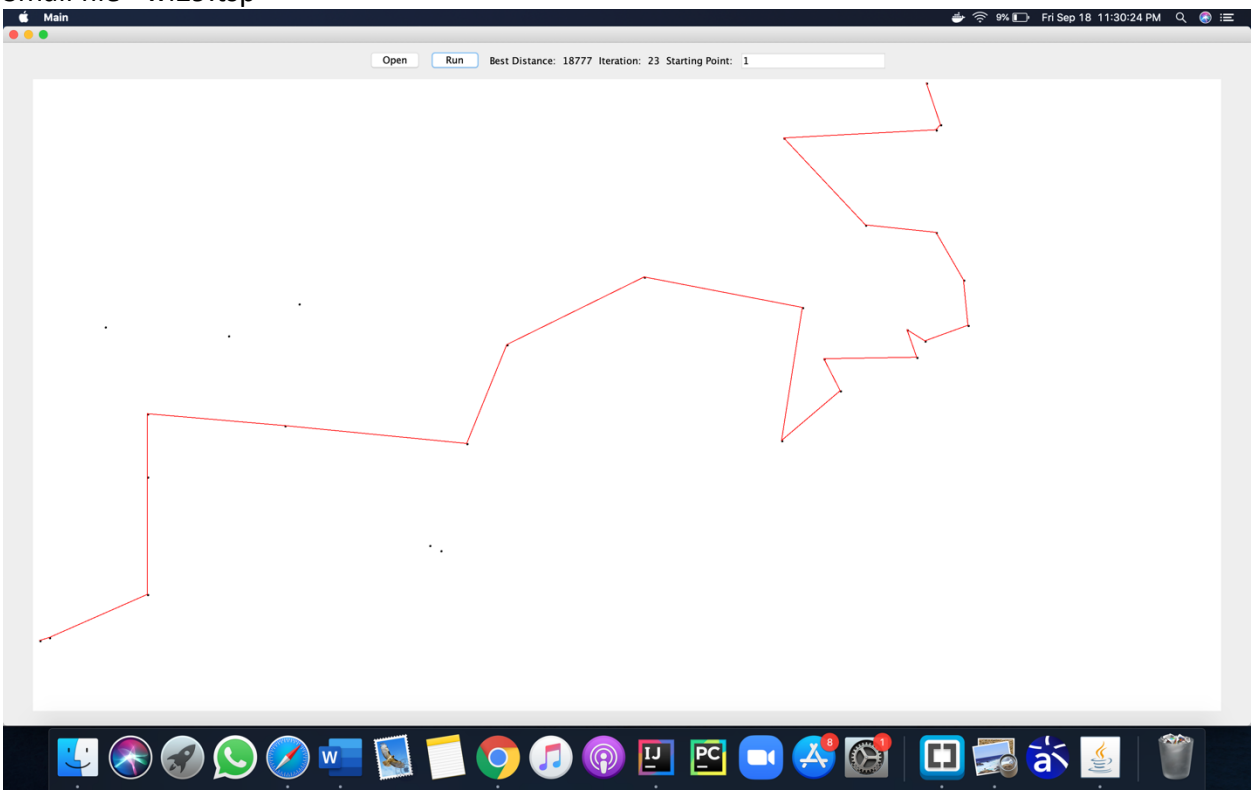
Separation of Concerns: (MVC)

Classes are divided based on the basis of its specific type of functionalities they perform. I have used Model View Controller design pattern to implement the solution. Model component represents the Java object that carries the data. Model component will be used by the controller component to update the data flow in the solution. In my solution the data is stored in the int array java object. Both DataExtractor and TSPSolution act as model in my solution. They update the state of the data which in turn updates the state of controller. The controller component in my solution controls the data flow. It interacts with both View and Model components in the design. It updates the view component based on the state of data. In my solution ButtonActionListener acts as the Controller component. It has functionality to open the file and use that data for calculating the TSP solution. It also updates the state in the view by updating the drawing panel based on the data points in the solution. The View component in this solution is the GUI that interacts with the user and used for visualization of the solution. SolutionGUI and

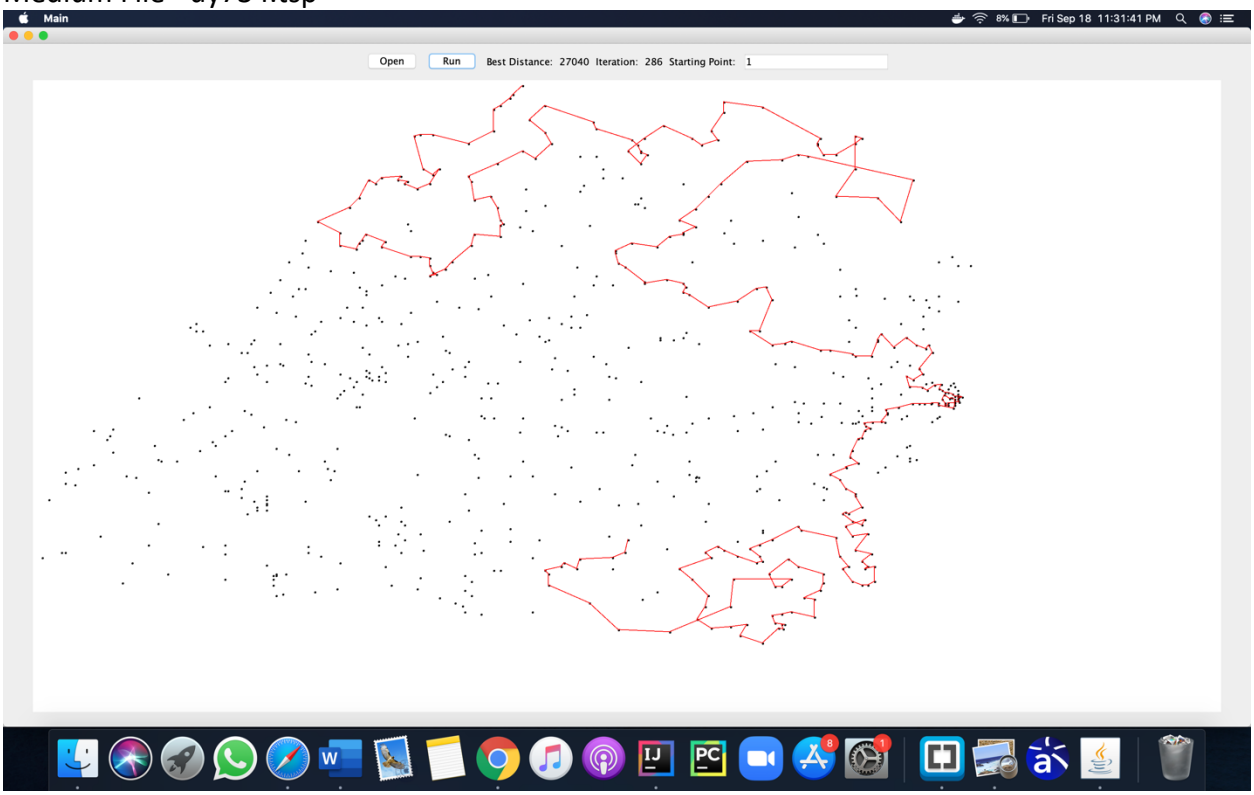
PlottinArea acts as the View component of the solution. They contain the JButton, JLabel and JPanel which constitute to the visualization of the solution.

Results:

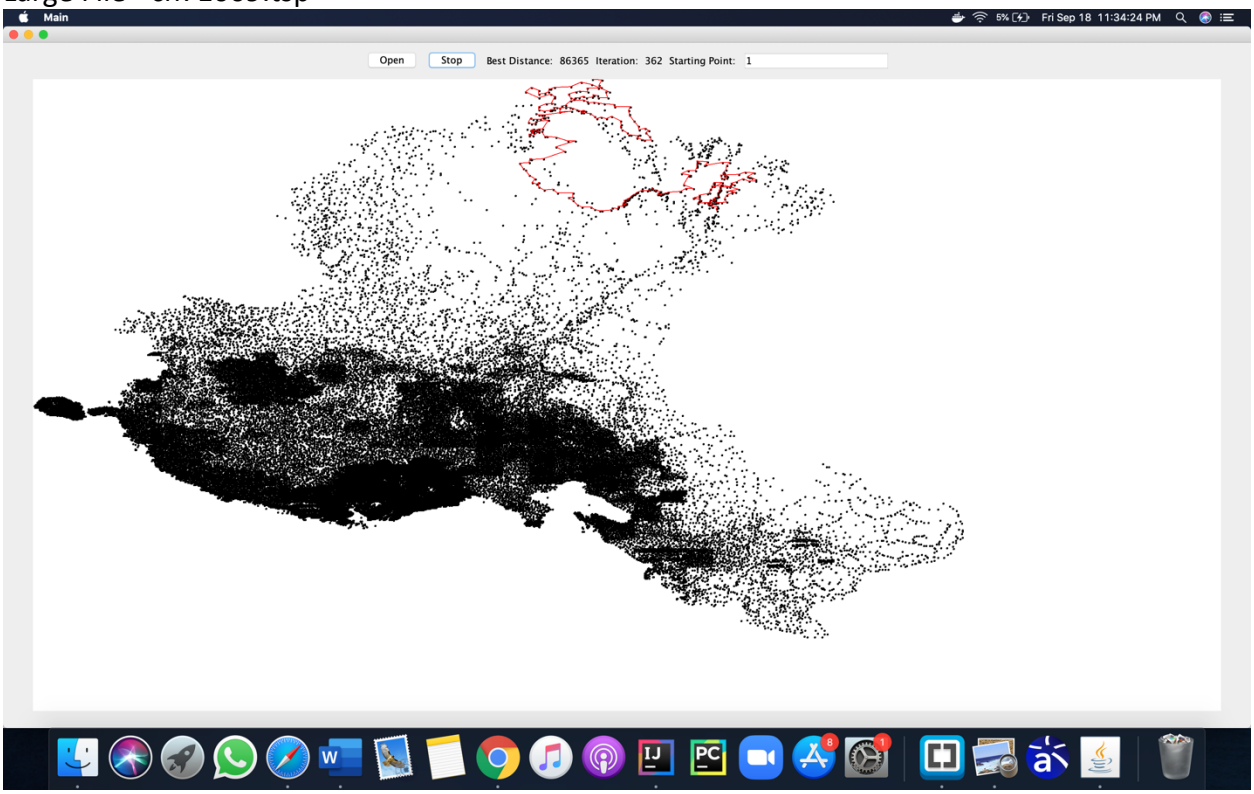
Small file - wi29.tsp



Medium File - uy734.tsp



Large File - ch71009.tsp



Design Decision:

I changed the way nearest neighbor algorithm is implemented so that the improved implementation runs even for very long data. Instead of calculating the distance between each city and storing it in a distance matrix, I calculated the distance only when that city is selected. By this way I was able to rectify the Java Heap size error I was facing in the previous assignment. I also decided to have a different drawing panel class instead of plotting the points and lines in the same frame. By using a different class for drawing panel I was able to efficiently repaint the panel. Also for each time I repaint the panel I maintain a set of variables based on which the content in the panel is decided. By separating out the different components I was able to implement a clean solution that is clean and easy to understand.

Conclusion:

Overall quality of the solution is good. I was able to implement all the requirements that are given by efficiently implementing the solution using good design pattern. My solution is refined and modularized at each level. I achieved low coupling by having less interdependency between each class. My solution was able to run all the data including large data like China with 71009 cities. I was able to plot all the points and connect the cities based on the route that is formed. Only limitation in my solution is that I compute all the routes for that given dataset and then start plotting the routes in the drawing panel. It can be improved by having a separate thread that computes the route and having another thread that plots the route that are computed this now. The class diagram that was generated by the Astah had some difference to the class diagram that I designed. In Astah the connections like compositions and aggregations are shown as associations. Class information and the associations between different classes were generated correctly.