Final Report: Automatic Generation of Transit Maps

John Lyon, BS CS

Advisor: Kenneth Blaha
Spring 2017

Abstract

When publishing subway maps, most transit agencies distort geography to more clearly show the topology of the network. This is accomplished by many techniques, including straightening lines, evenly spacing stations, and aligning elements to a grid. This project aims to automatically generate such a map from freely available data published by most transit agencies. Thus far I have been successful in loading the data and generating a geographically accurate map.

Contents

Table of Figures

1 Introduction

1.1 What is a Transit Map?

Most maps of subway, light rail, or similar transit systems use a simplified route schematic that sacrifices geographic accuracy to better show the interactions of the routes themselves. For example, consider how Google Maps presents the Washington, DC Metrorail system:

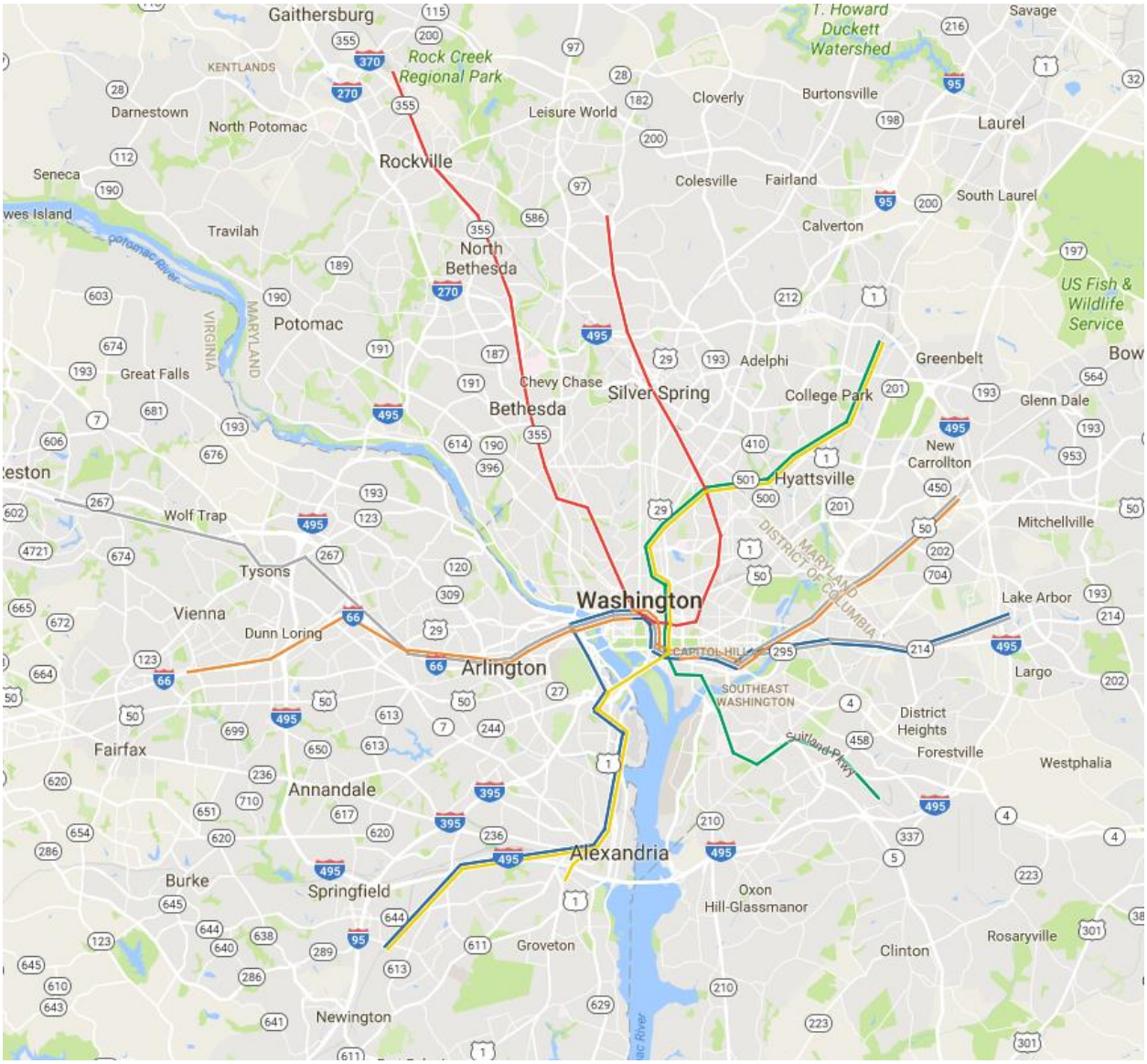

Figure 1.1-A: Google Maps representation of Washington, DC Metrorail system [1]

Now, compare this to the official system map:

Figure 1.1-B: Official Metrorail system map [2]

While the criteria that determine a great transit map is a matter of opinion, most follow a few key design points:

- Map elements are snapped to an underlying grid
- Squiggly route segments are simplified to straight lines
- The overall route is laid out with as few turns as possible
- Lines are aligned to standard angles (usually multiples of 45 degrees)
- Stations are evenly spaced along route lines
- The map retains an adequate relation to the physical geography

1.2 What is GTFS?

The General Transit Feed Specification (GTFS) was created by Google with the help of TriMet, the public transit agency in Portland, OR, for the purpose of importing transit route and schedule data to Google Maps. GTFS is an open data standard used by most transit agencies in the US and many around the world. GTFS data is downloaded from a transit agency's website in the form of a zip archive containing several row and comma separated text files. Data is stored across multiple files for various types of data (bus stops, routes, fares, etc.) and linked using key reference analogous to a relational database. Only a small portion of the data contained in a GTFS feed is used by this application. For more information on GTFS, visit the feed specification at https://developers.google.com/transit/gtfs/ [3]

1.3 Project Objectives

This project aims to automatically generate a transit map from an agency's GTFS feed.

1.3.1 Educational Goals

- Apply algorithms that are new to me
- Gain experience in developing medium-scale software projects
- Improve skills in project management
- Develop expertise in computer graphics generation

1.3.2 Functional Goals

- Load geographic data from a GTFS feed
- Allow user to specify lines and properties of a map
- Automatically generate a transit map that fits the specified criteria
- Display map to user and allow them to save to file

1.4 Project Results

Although the original aim of the project was to generate a schematic diagram similar to the one shown in Figure 1.1-A, at this point in time I have only managed to generate a

geographically accurate map similar to the one in Figure 1.1-B. Once generated this map can be displayed within the application but not saved to file.

2 Requirements

2.1 User Personae

In developing this project, I considered 2 positive user personae and one negative persona.

2.1.1 Primary Persona: Rebecca

Rebecca works in the public relations division of City Rapid Transit (CRT), a cash-strapped transit agency. CRT does not have the budget to hire a professional graphic designer to make them a route map, but needs a way to allow riders to navigate the system.

2.1.2 Secondary Persona: Cameron

Cameron is fascinated by the design of transit maps. He wants to quickly and easily compare multiple different maps of the same system with different parameters, in order to discover what makes an effective map.

2.1.3 Negative persona: Tim

Tim is a tired commuter who just needs to figure out how to get home at the end of his shift. Tim would be better off getting an official map from his transit agency than using this tool to generate one.

2.2 Design Limitations

2.2.1 Map Complexity

The difficulty of laying out a transit map increases significantly as more lines and stations are added. This is due to both the increased number of nodes to position and the increased likelihood of elements (particularly station labels) to overlap and form an unreadable mess.

2.2.2 Line Limitations

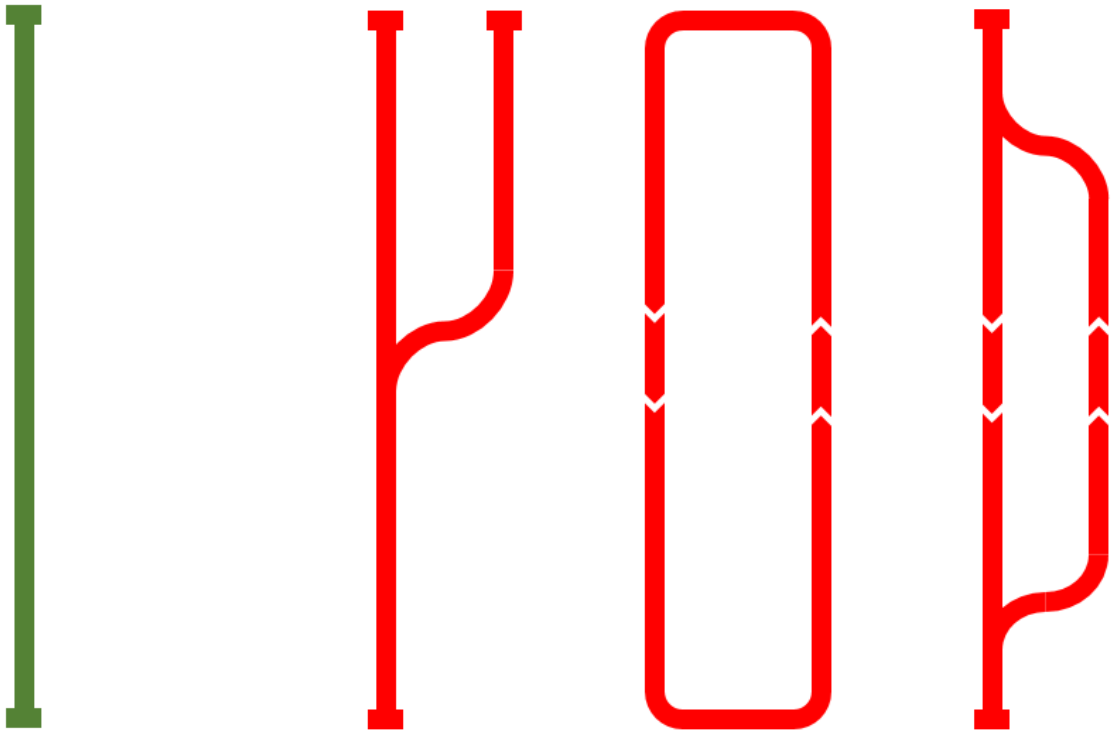

Figure 2.2-A: Acceptable and unacceptable line forms

Early in the project I decided on a limitation to make implementing the project more manageable: Each line on the map must be a simple line. Lines may not branch, loop, or have different routing in different directions.

2.3 User Stories

2.3.1 Choose File Location

> *As a mapping enthusiast, I want to choose which GTFS file to use to create a map, so I can have multiple GTFS files on my machine from multiple systems.*

Having the user specify a GTFS file to work on allows the program to run on a computer with multiple GTFS files, does not require the user to position the source file in a

specific location, and allows greater portability of the program. This user story has been successfully completed.

2.3.2 Load Data

As a mapping enthusiast, I want to load data from my local transit system's existing GTFS feed, so I can easily map real-world data.

Before a map can be created, the relevant data must be read from a GTFS feed. This user story has been successfully completed.

2.3.3 Select Lines

As a PR manager for a transit system, I want to select a subset of our lines to show on a map, so I can highlight our most important routes.

Many transit system's GTFS feeds contain a multitude of lines, both rail and bus. The type of generated by this project works better for a smaller subset of lines, and particularly rail lines. This user story has been successfully completed.

2.3.4 Select Trip for Lines

As a PR manager for a transit agency, I want to select a trip to represent each line, so I have more control of the final map.

To accommodate the design limitation discussed in section 2.2.2, each line is represented by a single trip. The line on the map will serve the stations served by that trip, in the order they are served by that trip. Allowing the user to pick the trip gives the user maximum flexibility when generating a map. This user story has been successfully completed. This user story may be rendered obsolete by future improvements the program, discussed in chapter 5.

2.3.5 Change Colors

As a PR manager for a transit agency, I want to specify the color of each line on the map, so as to match our branding.

Many transit agencies assign a color to transit lines as either a branding measure or to make a complicated map easier to read. These colors can be specified in the GTFS feed, but doing so is optional. The user can set the color of the line when it is not specified, or change the specified color to suit their taste. This user story has been successfully completed.

2.3.6 Tweak Names

As a mapping enthusiast, I want to change the names of lines and stations from that given in the GTFS feed, because the names in the GTFS feed do not always match the names displayed publicly.

The line and stop names given in a GTFS feed do not always match what is displayed publicly, or what one might desire to appear on a map. For example, King County Metro's RapidRide Line B is listed as stopping at "156th Ave NE & Overlake Transit Center – Bay 7" when just "Overlake Transit Center" might be a better map label. This user story has not yet been completed.

2.3.7 Change Optimization Criteria

As a mapping enthusiast, I want to change the parameters of the algorithm to better match each individual system, to produce the best map possible.

Any map layout is a tradeoff between multiple competing goals. To produce the best map, the program can allow the user to choose which measures to emphasize. Some possible choices include:

- Stations in straight lines
- Lines snapping to proper angles
- Regular station spacing
- Geographic accuracy
- Adherence to grid

This user story has not yet been completed.

2.3.8 Generate Optimized Map

As a PR manager for a transit agency, I want to generate a schematic transit map from our geographical GTFS data, so customers can better understand our transit system.

This is the core function of the application. After loading geographic data from a GTFS feed, the program will algorithmically lay out a schematic transit map of the type discussed in section 1.1. This user story has not yet been completed. As of writing, the program displays a geographically accurate map.

2.3.9 Display Map

As a PR manager for a transit agency, I want to quickly preview the generated map in the application itself, so I can check its quality before distributing to customers.

Upon generating the map, the simplest and most convenient way to present it to the user is to display it in the application window. This user story has been successfully completed.

2.3.10 Export Map

As the PR manager of a transit agency, I want to save the generated map in a format suitable to post on our website, so that it is available to customers.

Once a map has been generated it would be nice to save it as an image as opposed to only viewing it in the application window. If the map is saved as a vector image, such as a .svg file, it could be imported into a program such as Illustrator for further enhancement. This user story has not yet been completed.

3 Design

3.1 GTFS Data Structure

This data structure is used to store data read in from the GTFS file, and is structured similarly to the GTFS specification. This data structure also includes information gathered from the user about what is to be included in the final map.

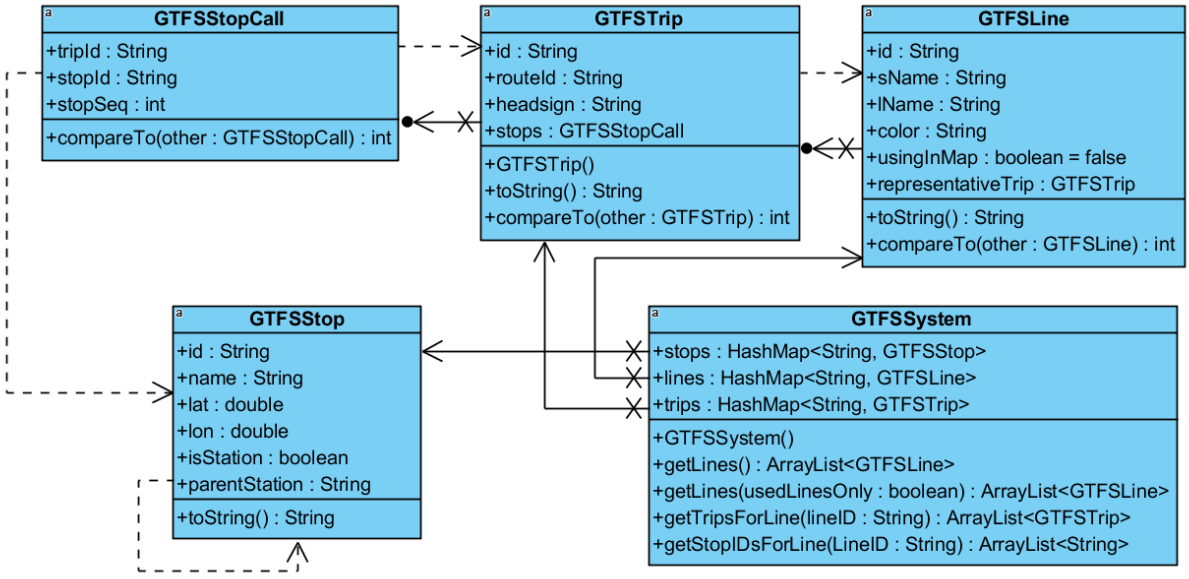

Figure 3.1-A: GTFS data structure design, for reading from file.

At the core of this data structure is the GTFSSystem class, which contains three hash maps. These maps use a String key (equal to the id field of the element) and a GTFSStop, GTFSLine, or GTFSTrip as the value. These equate to the concepts of Stop, Line, and Trip respectively (see glossary). These objects reference each other by storing a String key for the referenced object, which is looked up using the maps in the GTFSSystem object. This relation is shown on the diagram using dashed arrows. The GTFSStopCall class represents an instance of a transit vehicle (serving a trip) stopping at a stop. In the GTFS file this is used to store the time of a stop for scheduling purposes, but this program only uses it to connect the concepts of trips (and through them, lines) to the stations they serve. The GTFSLine class contains two fields, the Boolean value usingInMap and the GTFSTrip value representativeTrip, that store information gathered from the user.

3.2 SystemModel Data Structure

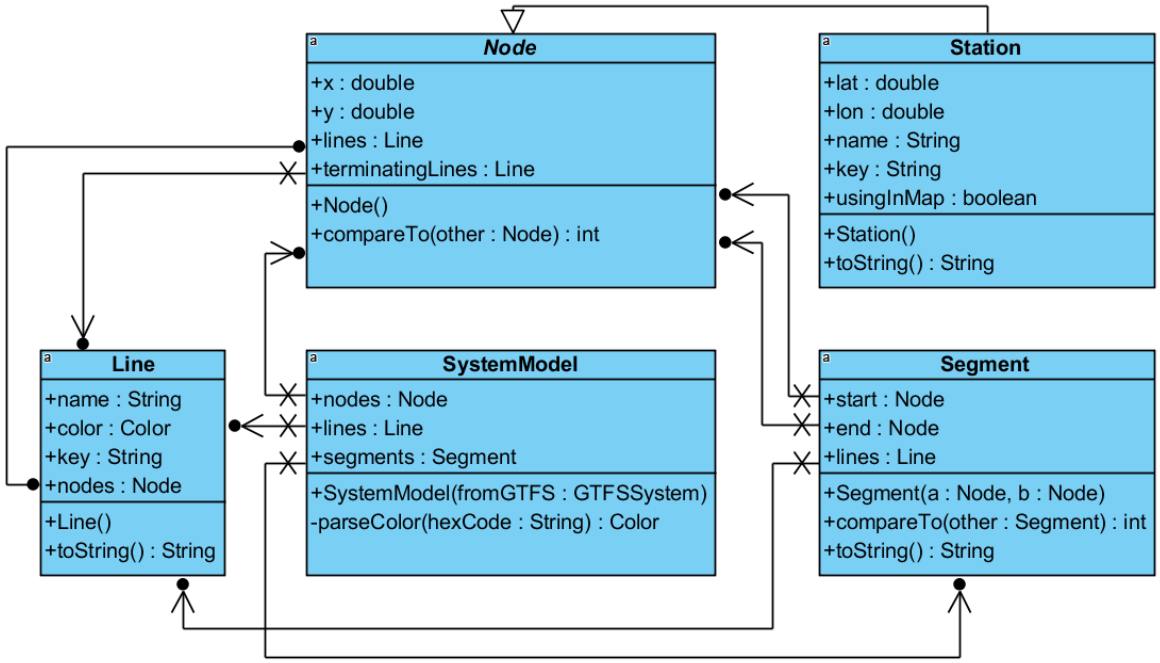

Figure 3.2-A: SystemModel Data Structure for storing, modifying, and drawing map data

This data structure is used to store the map data, including layout. Data is transformed from the GTFSSystem format to this after user input is complete. A few things happen as part of this transition:

- Lines and stops not selected to appear on the map are removed from the data structure
- Lines get directly mapped to stops using the representative trip selected in user story 2.3.4 (Select Trip for Lines).
- Coordinates are translated from real-world GPS values to an arbitrary coordinate space defined in the SystemModel class.
- Colors are translated from String representations to the Java Color class.

The base of this data structure is the SystemModel class, which contains ArrayLists of the other elements. The Line class is self-explanatory, and the Node abstract class represents a location on the map through which a line passes. At this time the only implementation of Node is Station (also self-explanatory) but in the future I plan to add a Corner type, to allow lines to change direction while not at a station. Lines and

Nodes reference each other directly, but do not at this time reference the third element of the data structure, the Segment. A Segment represents two adjacent Nodes, and lists every Line serving the segment. This allows segments served by two or more lines to be rendered properly, and also presents the future possibility of allowing branching lines (see section 5.3).

3.3 Program Flow

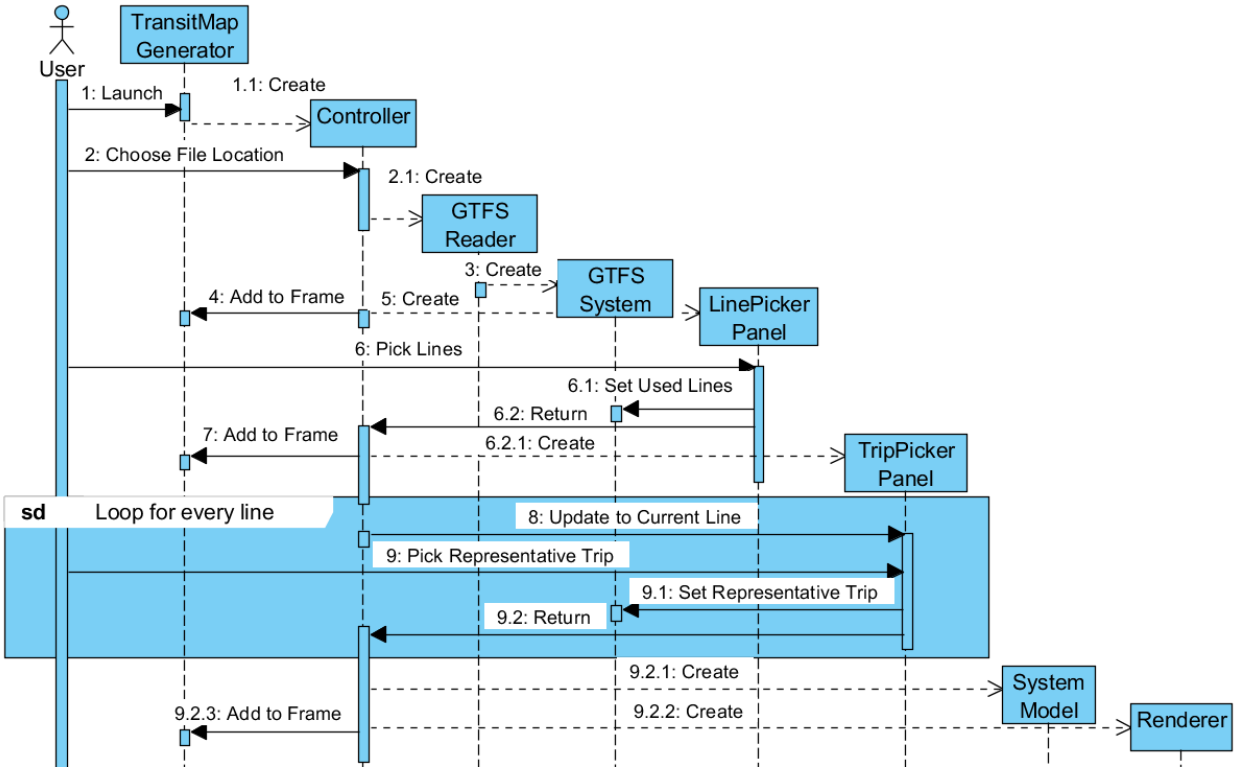

Figure 3.3-A: Program flow diagram

This diagram shows the execution flow of the program. Upon program launch, the TransitMapGenerator class (an extension of JFrame) creates the main window and a Controller object, which will manage the rest of the program. After the user picks a file location a GTFSReader is created to manage the data load, which creates the GTFSSystem data structure (see section 3.1). The Controller creates a LinePickerPanel (LPP, see section 3.4.1) and adds it to the main window, and the user uses it to select the lines to include on the map. The LPP updates the GTFSSystem accordingly and calls back to the Controller, which creates a TripPickerPanel (TPP, see section 3.4.2) and adds it to the main window, replacing the LPP. The Controller sets the active line in the TPP

and the user uses it to pick a representative trip and color for the line. The TPP updates the GTFSSystem accordingly and calls back to the Controller, which repeats this process for every line picked by the user and creates a SystemModel (see section 3.2) using the data from the GTFSSystem. The Controller then creates a Renderer (see section 3.4.3) and adds it to the main window. The program will endure in this state until closed by the user.

3.4 GUI Design

There are four main GUI panels used in the program, of which one is the Java stock file picking window. The remaining three are detailed below.

3.4.1 Line Picker Panel

This GUI implements user story 2.3.3 (Select Lines).

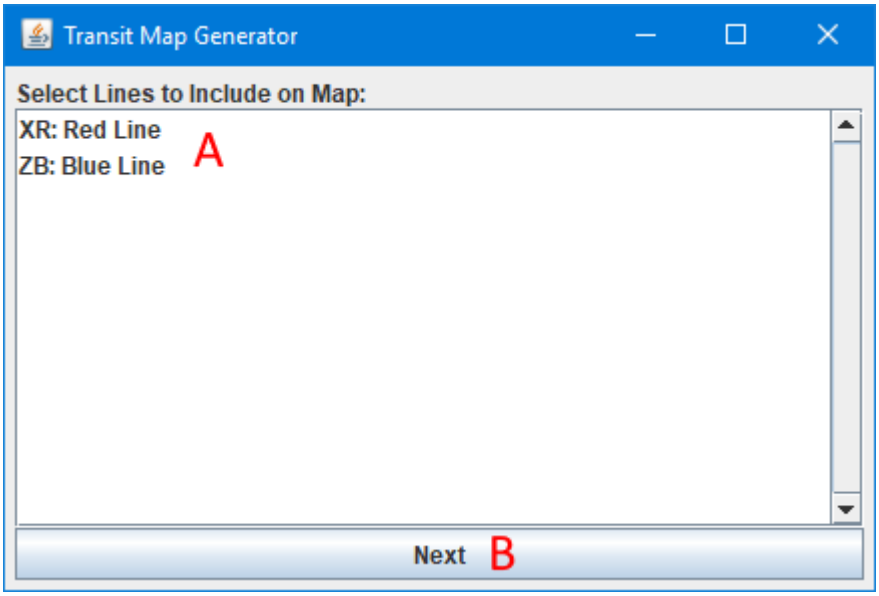

Figure 3.4-A: Line Picker Panel example

This is a very simple panel with only two main components:

A. This list box shows all transit lines included in the loaded GTFS feed. Any number of lines may be selected.
B. Clicking this button accepts the set of lines selected in list box A.

3.4.2 Trip Picker Panel

This GUI implements user stories 2.3.4 (Select Trip for Lines) and 2.3.5 (Change Colors).

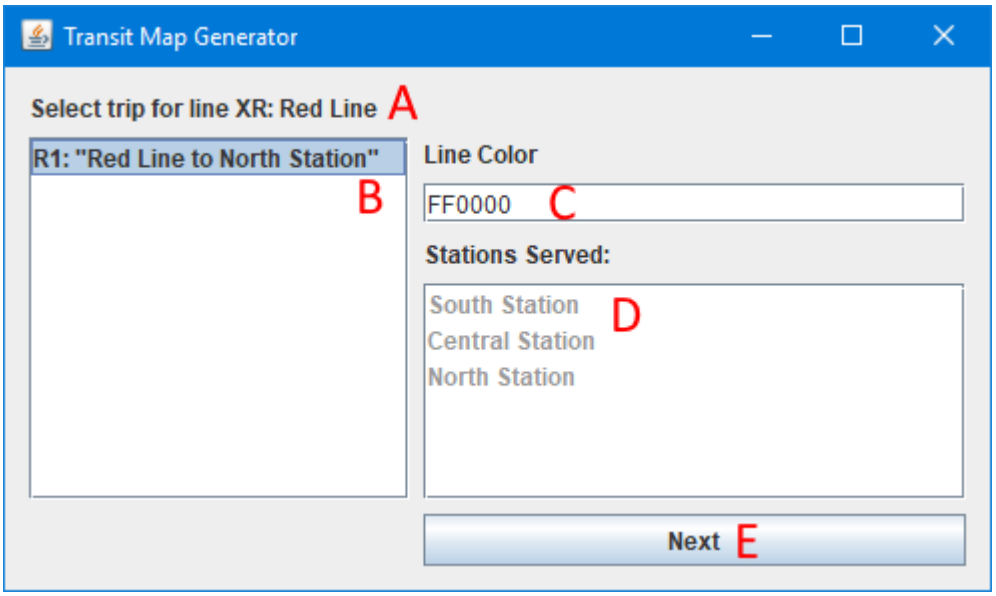

Figure 3.4-B: Trip Picker Panel example

This is a more complicated GUI with five main components. It is shown multiple times, once for each line selected to include on the map.

A. Label showing which line this instance of the GUI is working with
B. List box of all trips made by that line. Only one trip may be selected. The selected trip will be added as the representative trip of the line in label A.
C. Text box defining the color the line in label A will be drawn as in the map. If the line is assigned a color in the GTFS feed that color will show here as the default, otherwise the box will start blank. Colors are given as a standard 6-digit hexadecimal color code. Any input not valid as such will be given the default color black.
D. This list box shows the stops served by the trip currently selected in list box B. When the map is generated, the line in label A will serve exactly these stops in exactly this order.
E. Clicking this button accepts the trip selected in list box B and the color selected in text box C.

3.4.3 Rendering Panel

This panel renders the finished map for display to the user. It is not interactive.

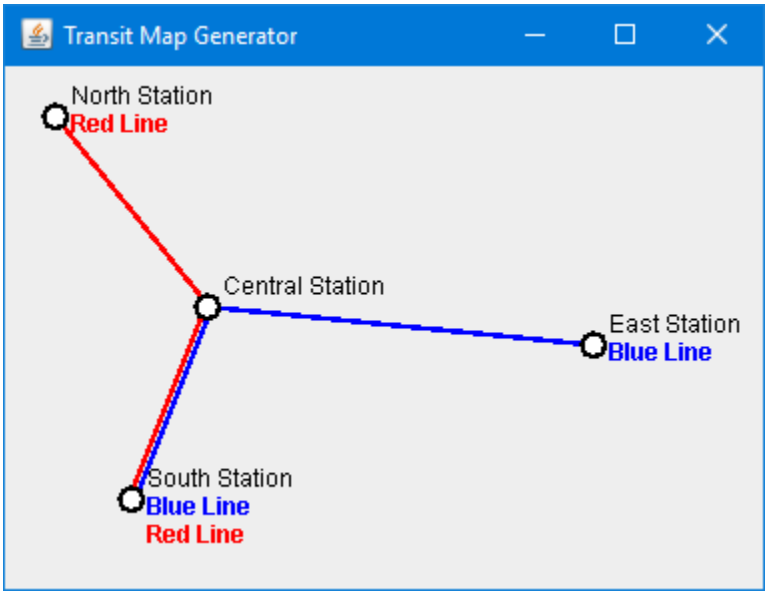

Figure 3.4-C Rendering Panel example

Maps are rendered according to the following rules:

- Stations are plotted according to the coordinates in the SystemModel data format
- Lines are drawn directly between stations
- Segments served by multiple lines will have all lines shown, evenly spaced and parallel to each other
- Stations appear as white circles with a black outline
- Station names are placed above and to the right of their respective station
- At terminus stations, the names of lines terminating are listed in bold text and line color below the station name
- Text is placed on top of station markers, which are placed on top of lines

4 Implementation

4.1 Tools and Process

This project was developed in Java using the Eclipse IDE. GUIs were developed using WindowBuilder, an Eclipse plugin. Project management was achieved using Trello, and code is stored in a GitHub repository. The project is coded almost entirely using stock Java APIs, but was originally planned to use two third-party libraries: A GTFS library from OneBusAway, an open source tool used to report scheduled and real-time public transit information, and Processing, a graphics tool. I was unable to figure out how to use the OneBusAway library, and eventually concluded that I could code my own reader for the minimal portion of GTFS data required to complete this project. Processing, meanwhile, was rejected once I discovered that Java has a stock graphics library adequate for my needs.

4.2 Challenges

I encountered several challenges when pursuing this project, many having to do with oddities in the GTFS format and individual transit agency's GTFS feeds. These are among the most vexing:

- The GTFS format contains a mechanism by which stops can be linked, such as saying that Westlake Station northbound platform and Westlake Station southbound platform are both part of the greater entity Westlake Station. This feature is optional, but my program relies on this data. Without it, this happens:

Figure 4.2-A: Doubled station in MARTA map

- In dense areas, text and even station markers can overlap, hindering readability to the point of uselessness. A better text rendering engine is needed to solve this problem, but for now I have added a mode where labels are left off, leaving a

map that is at least pretty, if not particularly useful.

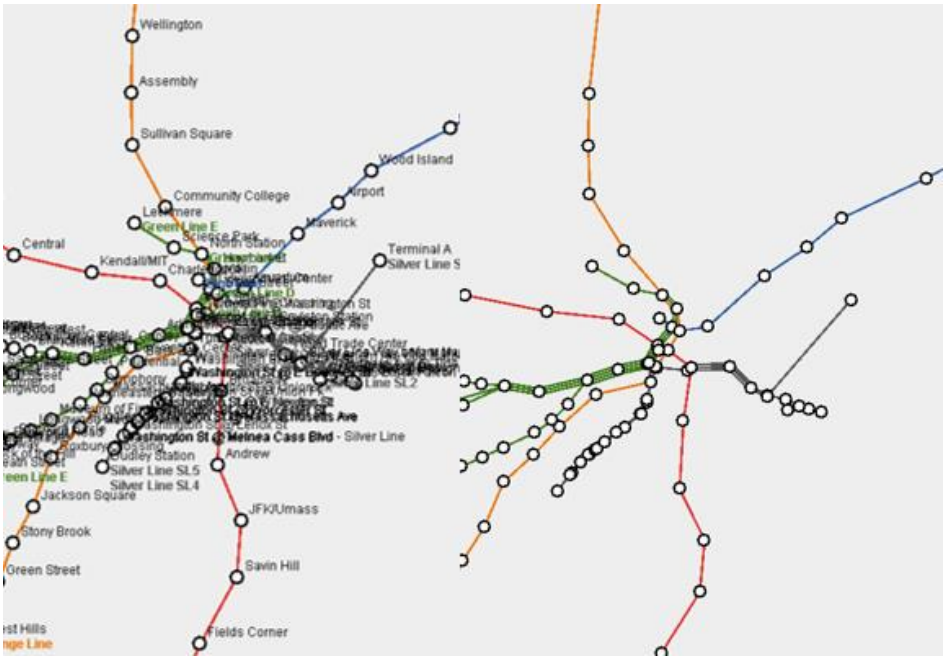

Figure 4.2-B: MBTA Rapid Transit with and without labels

4.3 Tested Systems

The program has been tested against GTFS feeds from the following transit systems and successfully produced maps (of varying quality):

- San Francisco BART
- Chicago "El" trains
- Denver RTD rail system
- King County Metro bus system
- Los Angeles Metro rail system
- Atlanta MARTA rail system
- Boston MBTA rapid transit and commuter rail systems
- New York Subway
- New York PATH train
- Pierce Transit bus system
- San Francisco MUNI Metro rail system
- Washington, DC Metro rail system

4.4 Sample Result

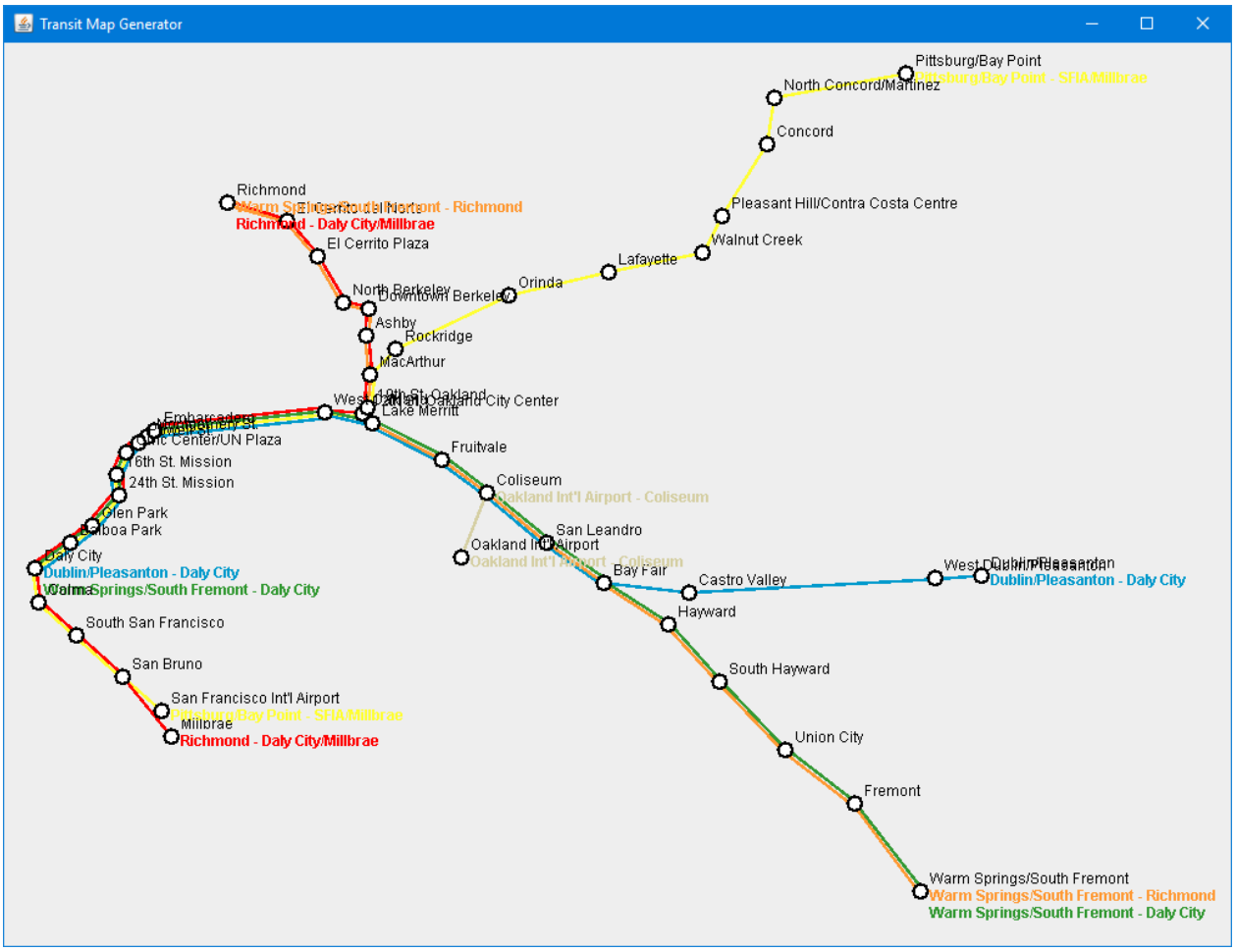

Figure 4.4-A: Generated BART map.

As an example of final output, this is the map generated by my program of the San Francisco Bay Area Rapid Transit system.

5 Future Work

5.1 Implement Layout Optimization Algorithm

By far the most important feature yet to be implemented is the optimization engine that transforms the geographically accurate layout currently generated into a stylized schematic diagram. A similar project has been attempted by Jonathan Stott for his PHD thesis, and example results are shown below.

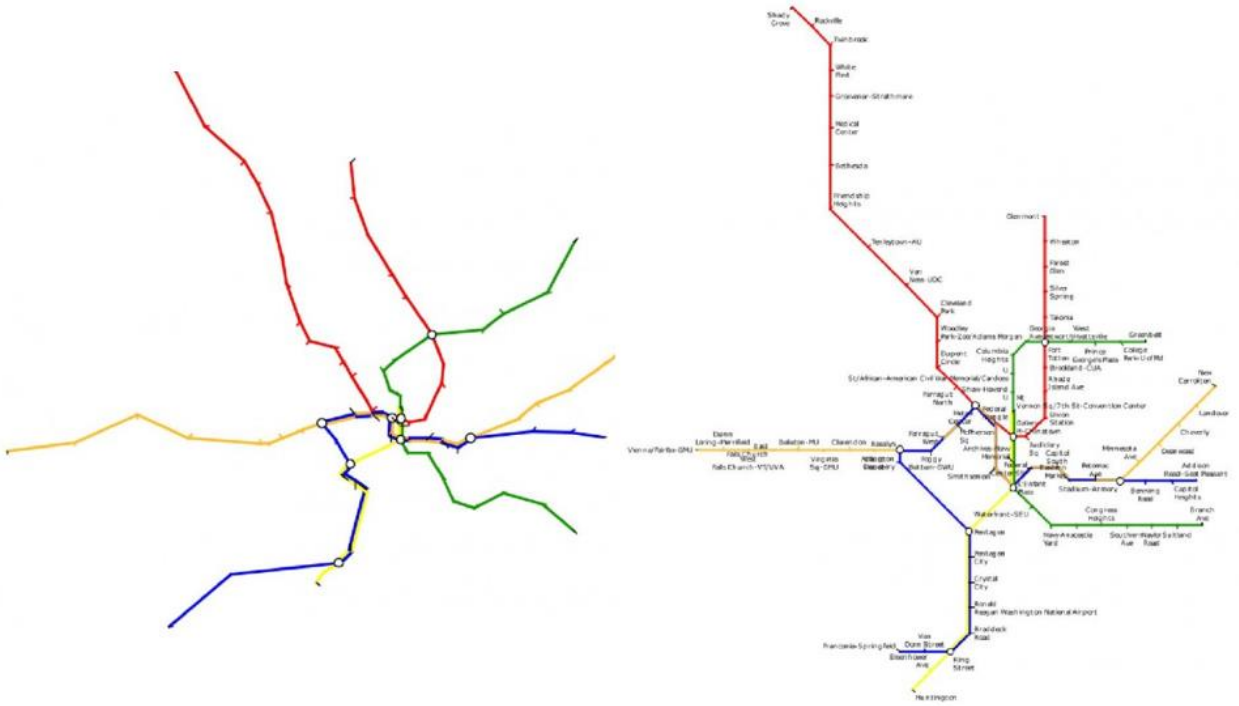

Figure 5.1-A: Washington Metro before (left) and after (right) optimization. [4]

Stott accomplishes his results using a multicriteria hill climbing algorithm, scoring maps on criteria such as the ones listed in User Story 2.3.7 (Change Optimization Criteria), then randomly shuffles the map and compares the new map's score to the old. It then picks the better map and repeats this process many times.

5.2 Improve Text Layout

Text layout has proved to be one of the more vexing problems in this project, and is one of the most difficult parts of designing any transit map. My current text placement system works well for lines that are diagonally oriented in a northwest/southeast

direction and with moderately wide gaps between stations, and poorly in most other contexts. Ideally I would like to implement a smart text placement system, possibly as part of the optimization engine discussed in section 5.1. Failing that, one idea I could implement would be to make the rendering engine interactive and show labels as a mouseover mode.

5.3 Allow Branching Lines

The SystemModel data format was changed very late in the project to include the Segment class, in order to allow the rendering engine to identify segments served by multiple lines and draw them accordingly. This change makes it feasible to store and render complex lines with multiple branching patterns, which are currently not supported by the program (see section 2.2.2).

5.4 Other Improvements

I have a wide variety of other ideas for new features or improvements, some of which are listed below.

- Automatically choose representative trips to remove the longest and most confusing user interaction with the software
- Replace the color text field in the Trip Picker Panel with a graphical color picker.
- Implement output to file
- Allow dynamic zoom of map in rendering panel

6 References

[1] Google, "Google Maps," 2017. [Online]. Available: https://www.google.com/maps/@38.9112922,-77.0149213,11z/data=!5m1!1e2. [Accessed 3 May 2017].

[2] Washington Metropolitan Area Transit Authority, "Metrorail System Map," 1 December 2015. [Online]. Available: https://www.wmata.com/schedules/maps/upload/system_map_color.pdf. [Accessed 6 October 2016].

[3] Google, "GTFS Static Overview," 26 July 2016. [Online]. Available: https://developers.google.com/transit/gtfs/. [Accessed 10 October 2016].

[4] J. M. Stott, "Automatic Layout of Metro Maps using Multicriteria Optimisation," University of Kent, Canterbury, 2008.

7 Glossary

- **GTFS**: General Transit Feed Specification, the data source for this project.
- **Line**: A transit line or route, such as the Piccadilly Line, the RapidRide Line B, or the Route 47 local bus. Can have a complex structure with multiple branches.
- **Map:** A graphical representation of a transit system.
- **Segment:** A set of two stops served directly in sequence, without intermediate stops.
- **Stop:** A location where you can board a transit vehicle.
- **Trip**: One trip on a line, with a transit vehicle leaving a station, eventually arriving at another station, and (usually) stopping at other stations in between. Strictly linear.