Interactive Train Schedules with String Charter

Gabriela Ožegović, Mauro Jurada, Elliot Koh, and Sven Celin

Information Visualisation SS 2020
Graz University of Technology
A-8010 Graz, Austria

29 Jun 2020

Abstract

This project reports explains and explores the tools, challenges and process of implementing the Ibry Train Schedule with String Charter.

Contents

List of Figures

Chapter 1

Introduction

The Ibry Train Schedule is string chart visualization that was invented as a way to visualize railway traffic. The aim of this project was to apply it for the ÖBB train system. This report will look at a short history of the Ibry Train Schedule and explore how it can be implemented on a modern system.

Chapter 2

Short History of the Ibry Train Schedule

Ibry Train Schedule is a graphic train schedule based on a line grid [Rendgen 2019]. It was Invented by Charles Ibry in the 1840s and it is an early visualization method for railway traffic (Figure 2.1). It is also known as a string chart. Edward Tufte, who focused on raising awareness of the history of information visualization, found the train schedule graphics in a seminal 19th century visualization book: Étienne-Jules Marey's "La méthode graphique."

The chart was invented so managers can easily see all trains on a given route over the course of a day. On the chart, train stops are inscribed on one of the axis, and on the other axis are inscribed hours of the day. Lines represent the trains themselves, and they connect the stops, which the train is going through, with regard to the time of the day when the train arrives at the specific stop. Since most of the early railway routes used a single track for driving in both directions, it was really easy to understand the graph and was an elegant way of representing them.

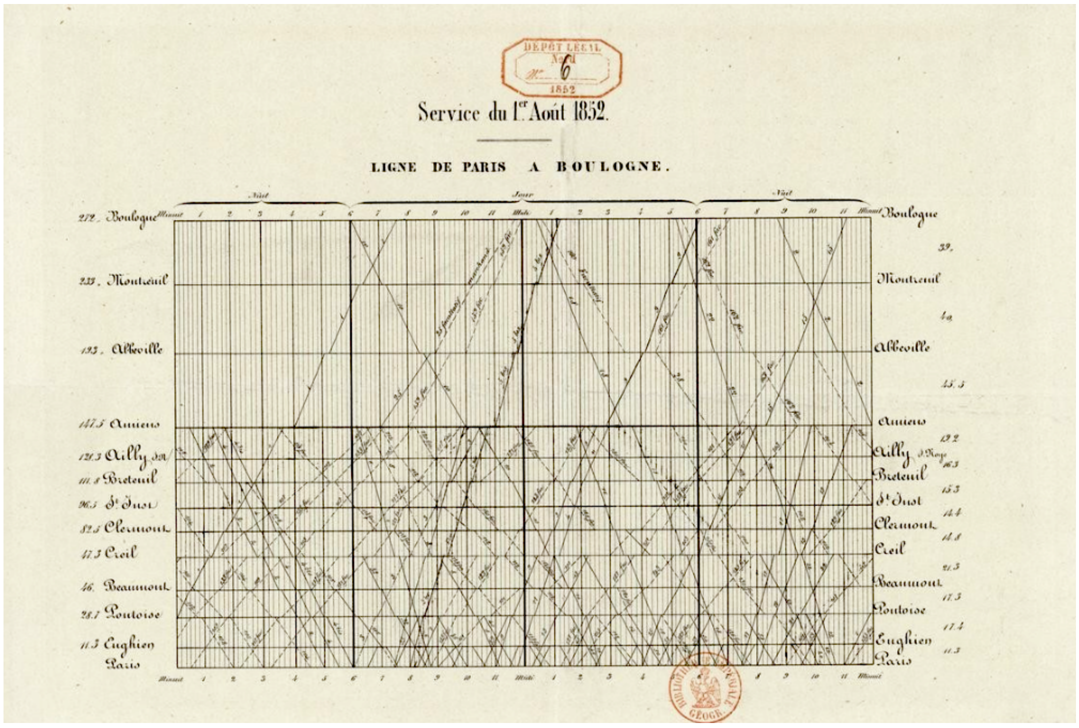


Figure 2.1: Ibry Train Schedule. [Ligne de Paris à Boulogne, 1852. Courtesy Bibliothèque nationale de France.]

Chapter 3

Tools

These are the tools used for making the String Charter.

3.1 Python (with libraries)

In this project Python[Python 2020] was used for making a script that makes a dataset that we will further use for drawing. For this data manipulation pandas library is used to make datasets and easily remove the columns that are not needed. Urllib is used for importing data from external link. Zipfile library is used for unzipping the data we get from the link. And in the end, we have used numpy for data manipulation and os for interacting with operating system (reading and writing).(Figure 3.1)

3.2 TypeScript

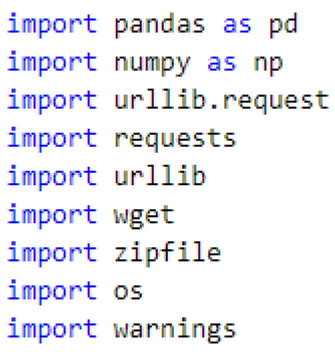
The project was built using TypeScript[Microsoft 2020].

3.3 D3.js

The D3.js[Bostock 2020] library was used to create and draw the String Chart.

3.4 Electron

Electron[Foundation 2020] is a framework for creating native applications with web technologies like JavaScript, HTML, and CSS. It was also used during the development phase for testing purposes.



```
import pandas as pd
import numpy as np
import urllib.request
import requests
import urllib
import wget
import zipfile
import os
import warnings
```

Figure 3.1: Imports for Python script. [Screenshot made by Sven Celin using Python script.]

Chapter 4

Data

The dataset used for this project is public data provided by OBB, national railway system of Austria. The dataset can be downloaded from their website. They provide data in both GTFS and NetEx format, and GTFS is the one used. The were some problems with the dataset, but that will be discussed in more detail in the Problems and Limitations chapter.

4.1 GTFS

GTFS (General Transit Feed Specification), also known as GTFS static or static transit, is a common format for public transportation schedules and associated geographic information developed by Google [Google 2020]. Using this format, public transit agencies can publish their transit data and developers can write applications that consume that data. It is a standardization of the public transport data. GTFS format consists of a series of at least six, and up to 13, text files, written in CSV file format, collected in a ZIP file. Each one of those TXT files contains particular transit information: agency, routes, trips, stop times, stops, calendar, and other, but optional, schedule data (Figure 4.1) . Preferred character encoding is UTF-8. It is designed to not only be able to provide trip planning functionality, but also to be able to provide analysis of the service and performance measures. It is limited to scheduled information, and does not use real-time information. For that reason, a GTFS realtime extension was developed, which can be added.

The main reason why the GTFS was developed was because Google wanted to incorporate transit into Google Maps. It was something that was needed, and after it was developed, a lot of developers were able to base their software on the format, and this resulted in a large number of useful and popular transit applications.

4.2 Data needed

To be able to draw the chart effectively, the dataset had to be combined and saved in a single CSV file. The expected format of the dataset is a table, where each row represents one trip (one train). The columns are stops, so for each row, the value in a specific column is the time at which the train arrives at that stop. This way, the times and stops for each train are easy to access. The approach to this task is discussed in the further chapter.

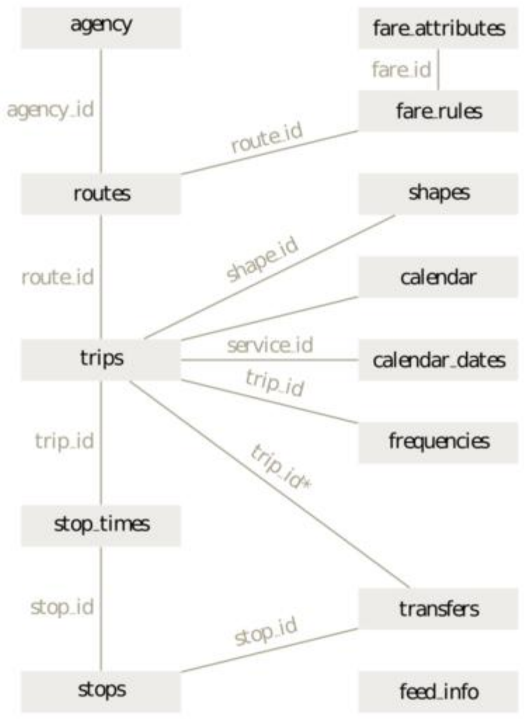


Figure 4.1: GTFS Class Diagram. [Published in “Opening Public Transit Data in Germany”.]

Chapter 5

Python script

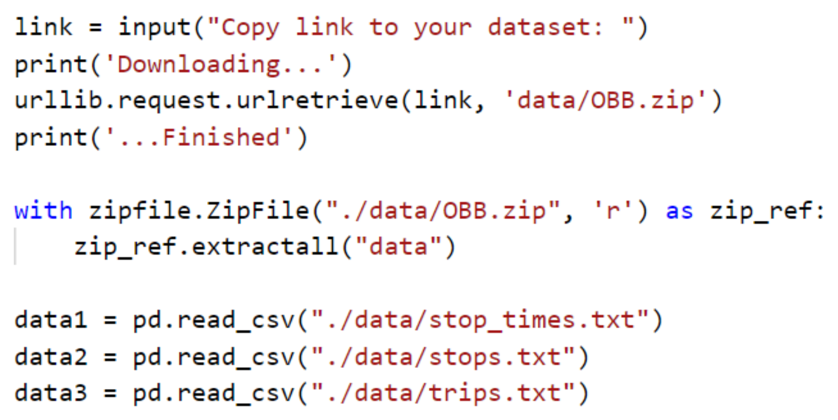
In this section, Python script will be discussed and all of its core mechanics and options it offers. Python script is the main tool for downloading CSV files from a given data provider, merging data needed for this projection, data preprocessing, listing stations, user choosing stations and, in the end, making the dataset that can be loaded in the frontend of the application. This script can be used on any standardized dataset that supports GTFS and it can produce a drawable dataset in the end.

5.1 Preprocessing

In this part of the script, the user is asked to provide the link to a dataset to be downloaded, unzipped and placed in a given folder for further use. As explained in the previous section, the given dataset is made with the help of GTFS standard and it has a lot of data that is not needed for this project. The script only needs "stop_times.txt", "stops.txt" and "trips.txt" for making the final dataset so the script only loads these three files (Figure 5.1). After loading necessary files script removes columns from the dataset that are not used and saves the dataset as CSV in file "all_routes.csv". Script then continues to put all names of the stops in terminal window for user to see. After making this list user is asked to input two stops that will become first and final stop for the final dataset. After inputting the data we need, the script continues to advance to filter only trips with these two stops. After finding only these trips it advances to delete all the cities before first stop and after last stop to have only stops that user inputs and all in between. Making this dataset it is almost finished. Final thing to do is only to see what order the stops are having in the dataset and this is made from time they arrive to the given stop (Figure 5.2). In this way there is no bias that could happen when handling "stop_sequence" because sequences of the stops are not always the same or in some routes trains could skip a stop and it won't be usable. In this way ordering of the train stations are perfectly ordered by time. Final stop of handling the dataset is to make it in format that is needed for drawing (Figure 5.3). This formatting is done in the final step and it ordered the trips by row and all necessary data is described in columns. Columns that are needed are "trip_id", "direction", "trip_short_name" and list of stops depending on route.

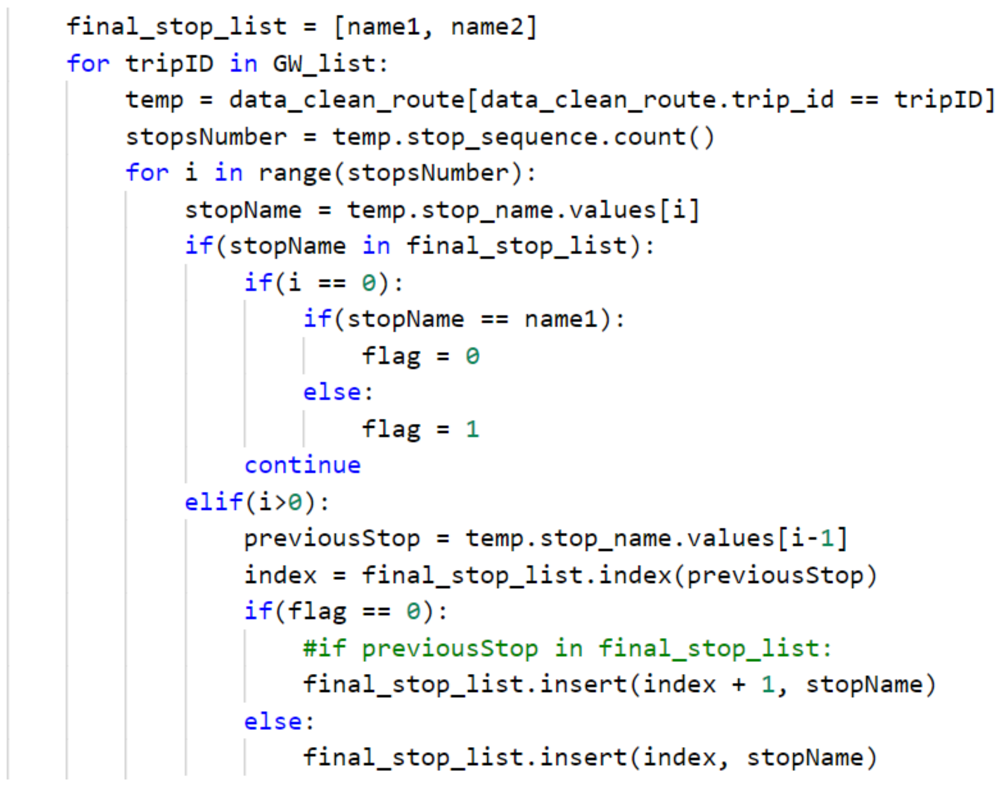
5.2 User Interaction

User interacts with the script on three occasions. When he inputs a link to a dataset of their preferred railroad transporting system. Second is when the user inputs the starting city from which he wants to start traveling and third is when the user inputs the final stop of his trip (Figure 5.4).



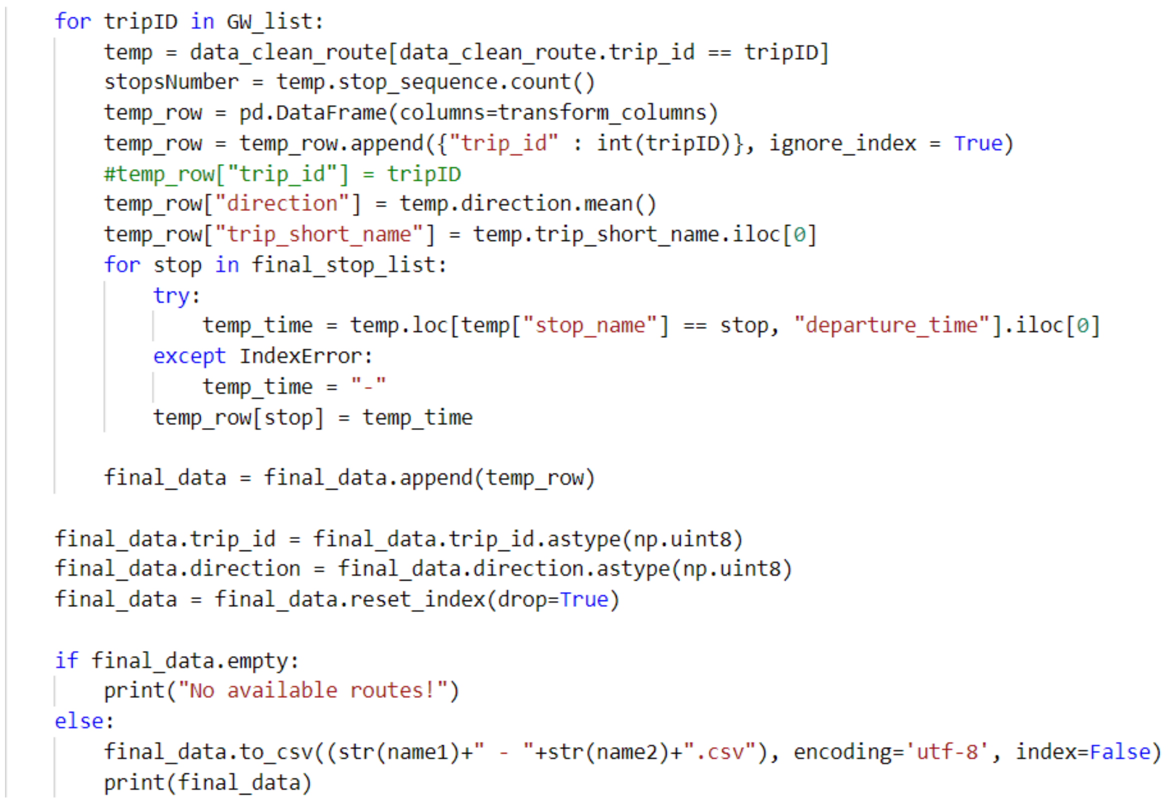
```
link = input("Copy link to your dataset: ")
print('Downloading...')
urllib.request.urlretrieve(link, 'data/OBB.zip')
print('...Finished')

with zipfile.ZipFile("./data/OBB.zip", 'r') as zip_ref:
    zip_ref.extractall("data")

data1 = pd.read_csv("./data/stop_times.txt")
data2 = pd.read_csv("./data/stops.txt")
data3 = pd.read_csv("./data/trips.txt")
```

Figure 5.1: Inport of the initial dataset. [Screenshot made by Sven Celin using Python script.]



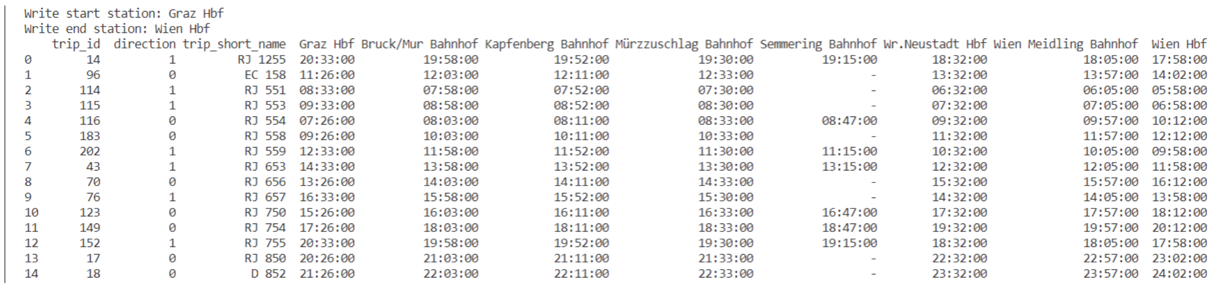
```
final_stop_list = [name1, name2]
for tripID in GW_list:
    temp = data_clean_route[data_clean_route.trip_id == tripID]
    stopsNumber = temp.stop_sequence.count()
    for i in range(stopsNumber):
        stopName = temp.stop_name.values[i]
        if(stopName in final_stop_list):
            if(i == 0):
                if(stopName == name1):
                    flag = 0
                else:
                    flag = 1
            continue
        elif(i>0):
            previousStop = temp.stop_name.values[i-1]
            index = final_stop_list.index(previousStop)
            if(flag == 0):
                #if previousStop in final_stop_list:
                final_stop_list.insert(index + 1, stopName)
            else:
                final_stop_list.insert(index, stopName)
```

Figure 5.2: Function for ordering the stops. [Screenshot made by Sven Celin using Python script.]



```python
for tripID in GW_list:
    temp = data_clean_route[data_clean_route.trip_id == tripID]
    stopsNumber = temp.stop_sequence.count()
    temp_row = pd.DataFrame(columns=transform_columns)
    temp_row = temp_row.append({"trip_id" : int(tripID)}, ignore_index = True)
    #temp_row["trip_id"] = tripID
    temp_row["direction"] = temp.direction.mean()
    temp_row["trip_short_name"] = temp.trip_short_name.iloc[0]
    for stop in final_stop_list:
        try:
            temp_time = temp.loc[temp["stop_name"] == stop, "departure_time"].iloc[0]
        except IndexError:
            temp_time = "-"
        temp_row[stop] = temp_time

    final_data = final_data.append(temp_row)

final_data.trip_id = final_data.trip_id.astype(np.uint8)
final_data.direction = final_data.direction.astype(np.uint8)
final_data = final_data.reset_index(drop=True)

if final_data.empty:
    print("No available routes!")
else:
    final_data.to_csv((str(name1)+" - "+str(name2)+".csv"), encoding='utf-8', index=False)
    print(final_data)
```

Figure 5.3: Function for making final look of the dataset. [Screenshot made by Sven Celin using Python script.]



```
Write start station: Graz Hbf
Write end station: Wien Hbf
    trip_id  direction  trip_short_name  Graz Hbf  Bruck/Mur Bahnhof  Kapfenberg Bahnhof  Mürzzuschlag Bahnhof  Semmering Bahnhof  Wr.Neustadt Hbf  Wien Meidling Bahnhof  Wien Hbf
0        14          1          RJ 1255  20:33:00           19:58:00            19:52:00              19:30:00           19:15:00         18:32:00               18:05:00  17:58:00
1        96          0           EC 158  11:26:00           12:03:00            12:11:00              12:33:00                  -         13:32:00               13:57:00  14:02:00
2       114          1           RJ 551  08:33:00           07:58:00            07:52:00              07:30:00                  -         06:32:00               06:05:00  05:58:00
3       115          1           RJ 553  09:33:00           08:58:00            08:52:00              08:30:00                  -         07:32:00               07:05:00  06:58:00
4       116          0           RJ 554  07:26:00           08:03:00            08:11:00              08:33:00           08:47:00         09:32:00               09:57:00  10:12:00
5       183          0           RJ 558  09:26:00           10:03:00            10:11:00              10:33:00                  -         11:32:00               11:57:00  12:12:00
6       202          1           RJ 559  12:33:00           11:58:00            11:52:00              11:30:00           11:15:00         10:32:00               10:05:00  09:58:00
7        43          1           RJ 653  14:33:00           13:58:00            13:52:00              13:30:00           13:15:00         12:32:00               12:05:00  11:58:00
8        70          0           RJ 656  13:26:00           14:03:00            14:11:00              14:33:00                  -         15:32:00               15:57:00  16:12:00
9        76          1           RJ 657  16:33:00           15:58:00            15:52:00              15:30:00                  -         14:32:00               14:05:00  13:58:00
10      123          0           RJ 750  15:26:00           16:03:00            16:11:00              16:33:00           16:47:00         17:32:00               17:57:00  18:12:00
11      149          0           RJ 754  17:26:00           18:03:00            18:11:00              18:33:00           18:47:00         19:32:00               19:57:00  20:12:00
12      152          1           RJ 755  20:33:00           19:58:00            19:52:00              19:30:00           19:15:00         18:32:00               18:05:00  17:58:00
13       17          0           RJ 850  20:26:00           21:03:00            21:11:00              21:33:00                  -         22:32:00               22:57:00  23:02:00
14       18          0            D 852  21:26:00           22:03:00            22:11:00              22:33:00                  -         23:32:00               23:57:00  24:02:00
```

Figure 5.4: User interaction with application. [Screenshot made by Sven Celin using Python script.]

5.3 Conclusion

This script makes any GTFS dataset usable in this application. If the dataset and the link is made in boundaries of the GTFS specification it will always make usable dataset for use in the visual part of the application.

Chapter 6

Features

In this section, the core features and functionality of String Charter is discussed. String Charter was developed to allow users to customize their experience to fit their needs. In terms of cutomizability, users have the option filter trains by time and direction as well as change the colour of the chart. Users can hover over stations and stops that interest them to get more information. Users also can export their customized chart as a SVG image.

6.1 Import from CSV

As explained in the data section, String Charter accepts data as a CSV file (Figure 6.1) in the format described above. Users have the option to use the script provided or create their own CSV data file.

6.2 Filter by Direction

The ability to filter by direction of travel (Figure 6.2) is useful. Users who are interested in only one direction can remove clutter by filtering irrelevant information. On the other hand, users who are interested in visualizing the entirety of the train schedule can do so.

6.3 Filter by Time

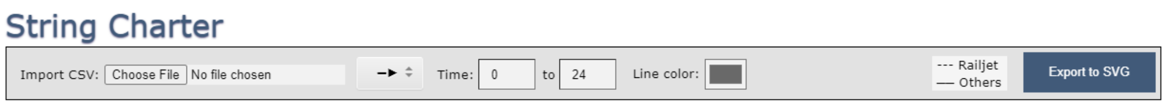
The ability apply a time range filter (Figure 6.1) is also useful. Users can select which times they are interested in viewing.

6.4 Line Colours

Users have the option to select the colour of the lines used. This is especially helpful when visualizing a schedule with many trips in both directions (Figure 6.3). Using a different colour for both directions is useful in being able to distinguish trains going in different directions more quickly. By default, the second line is set to the contrasting colour of the first line.

6.5 Information on Hover

Users can hover over stops to get more information(Figure 6.4). The tooltip provides the train name, station name as well as the exact time the train arrives at the station.



String Charter

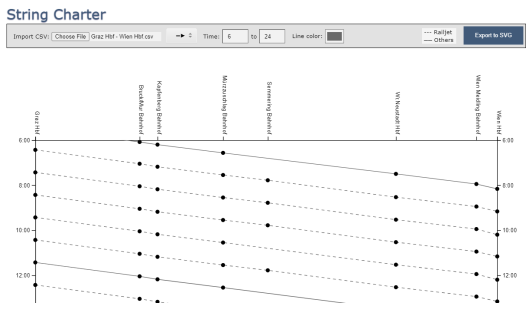
Import CSV: Choose File No file chosen

Time: 0 to 24 Line color:

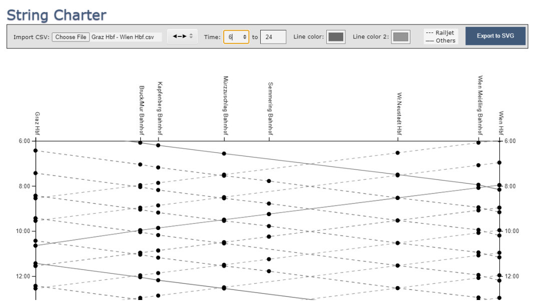
--- Railjet
— Others

Export to SVG

Figure 6.1: String Charter's menu bar. [Screenshot made by Elliot Koh using String Charter.]

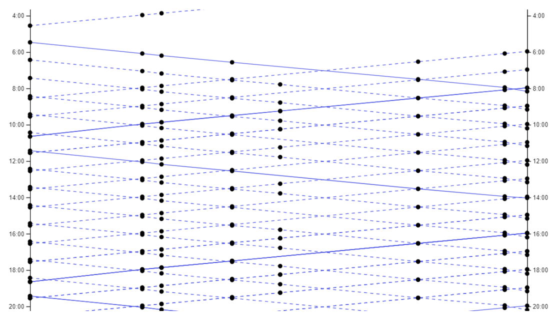


(a) Filtered to show only trains going from "Graz Hbf to Wien Hbf".

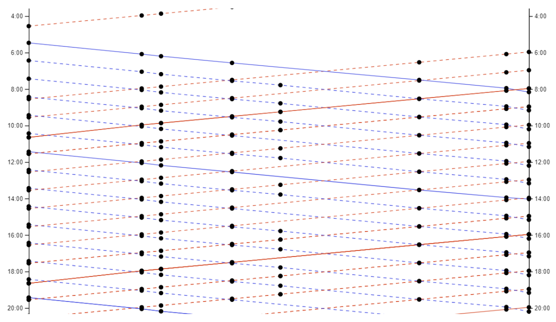


(b) Filtered to show trains going in both directions.

Figure 6.2: Trips can be filtered by direction of travel. [Both screenshots made by Elliot Koh using String Charter.]



(a) Drawing all trips with the same colour.



(b) Drawing trips in opposite directions with the contrasting colours.

Figure 6.3: Users are free to select how lines are coloured. [Both screenshots made by Elliot Koh using String Charter.]

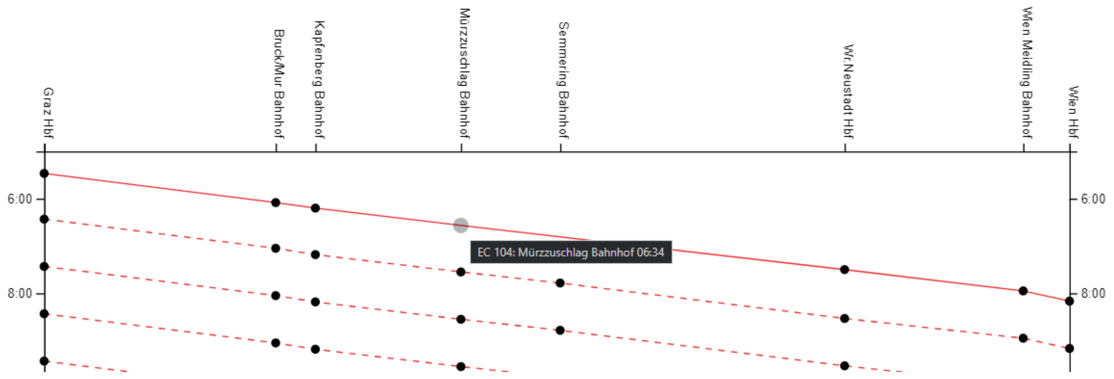


Figure 6.4: Information on hover. [Screenshot made by Elliot Koh using String Charter.]

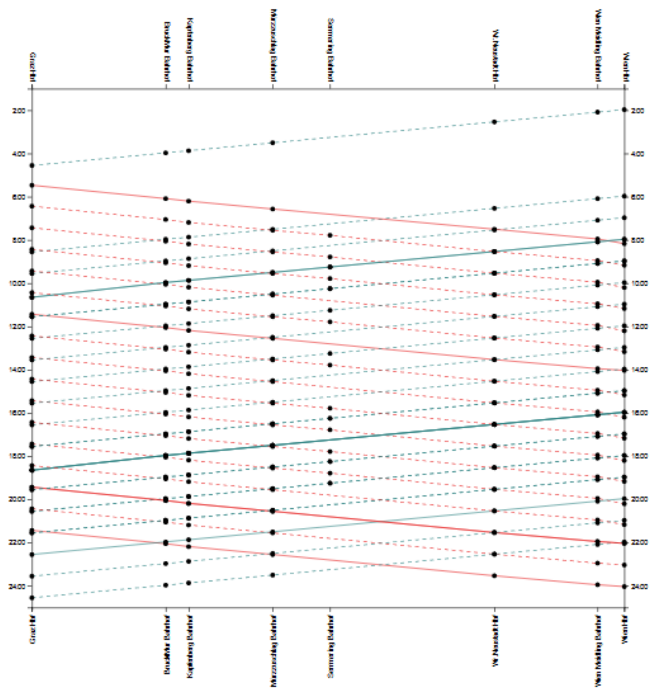


Figure 6.5: Exported SVG image. [Screenshot made by Elliot Koh using String Charter.]

6.6 Export as SVG

Users can export the customized chart as an SVG image. The SVG image uses a viewbox and as a result scales with the size of its parent container (Figure 6.5).

6.7 Conclusion

The features of String Charter were developed with convenience and customizability in mind. This was achieved by allowing users to customize the chart to their needs and preferences. The export feature also lets users use their created chart wherever they choose.

Chapter 7

Project Iterations

Progress from a basic mockup to the final version of the project will be shown in this chapter.

7.1 Mockup

Before starting any programing, a UI design was collaboratively drawn up using MockFlow[MockFlow 2020] as seen in the Figure 7.1. Using this mockup allowed us to easily understand and envision future progress.

7.2 Finalising the Project

There were a number of iterations, but the most important change from the mockup was the decision to switch up the content location by putting the menu bar on top. This allows for more screen room for the string chart.

Secondly, we can now see how it looks (Figure 7.2) and by testing numerous datasets, we realised which filter options work and which do not. The "Flip Chart" button didn't make sense anymore and the dataset didn't differentiate the weekdays from weekends. There were numerous improvements on the actual String Chart drawing on every step of the way.

Finally, in the Figure 7.3 the menu was cleaned up and filled with features already discussed in the previous chapter. Choosing the time 0 to 0 removes all the whitespace from the chart. Switching to show both directions adds another color palette if the user wants to modify the automatically picked complementary colour. There is also a legend describing the routes.

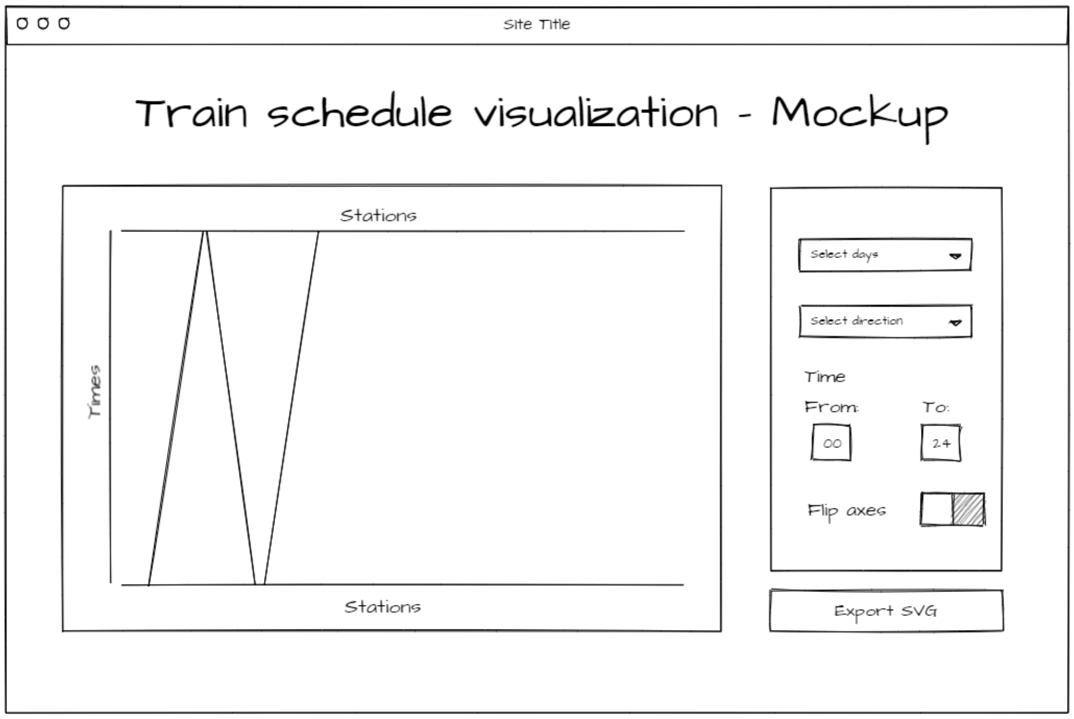


Figure 7.1: First mockup. [Screenshot made by Mauro Jurada using MockFlow.]

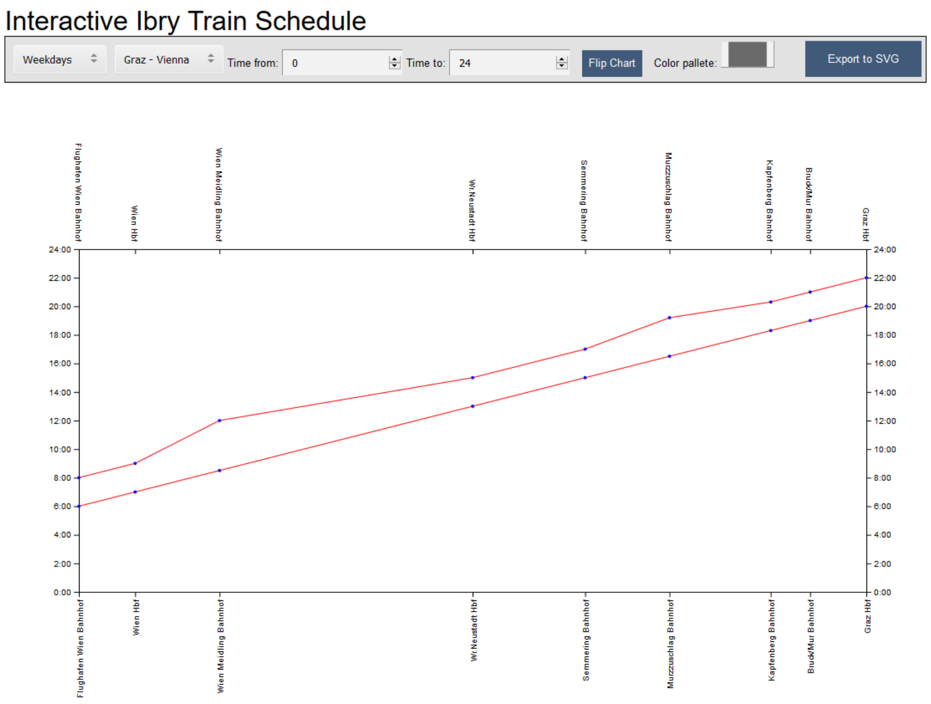


Figure 7.2: String Charter's first prototype. [Screenshot made by Mauro Jurada.]

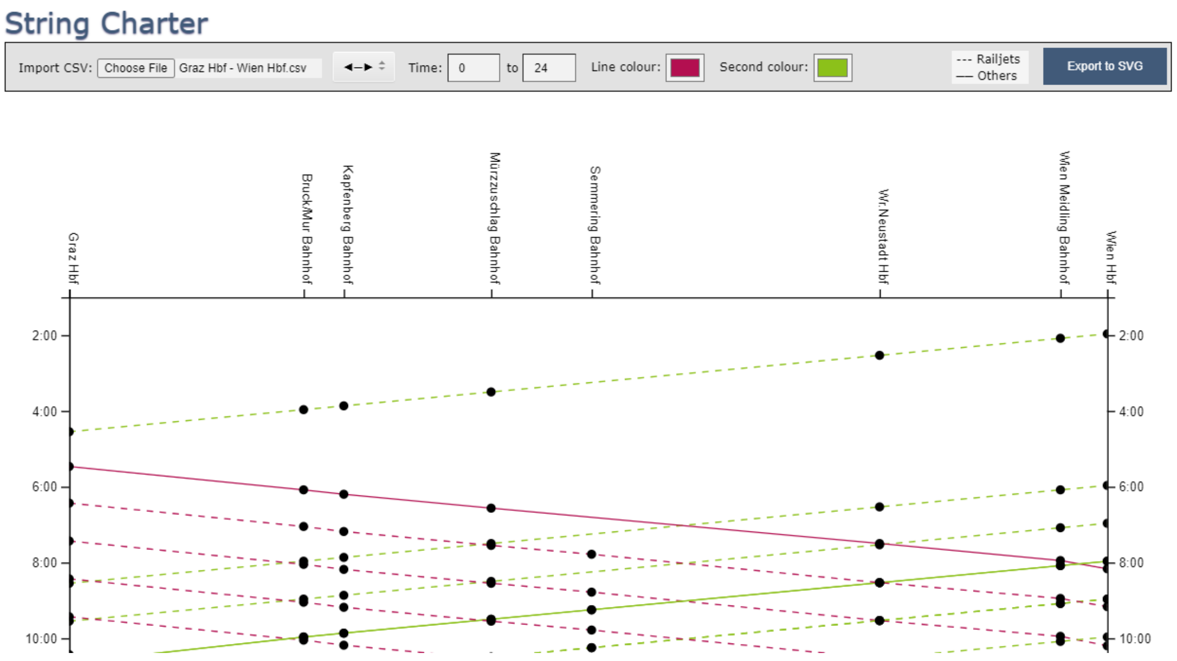


Figure 7.3: String Charter's final version. [Screenshot made by Mauro Jurada using String Charter.]

Chapter 8

Problems and Limitations

While working on this project, multiple problems were faced. These problems will be discussed in further sections.

8.1 Dataset problems

One of the biggest problems was the dataset. Automatically fetching the data via the URL, so it updates with each change from the OBB's side, was not possible. Firstly, to open the website which contains the data, a user should select a certificate, which is done for security reasons. To try and avoid this issue, we found the same dataset on the Austrian Government's website, and tried to use that to fetch the data. Unfortunately, this also did not work, since the URL itself would not lead to the file directly, but to some other script, and there were several steps between the link and the data itself. Because of this, we had to download the data manually.

The second problem we faced with the dataset is the fact that it was incomplete. For example, the calendar table, which is stored in calendar.txt file, was just filled with zero values, and nothing else. This limited us and we could not implement a functionality to choose a specific day and see the trips for that day, which was our idea at the beginning.

8.2 Listing the routes

To make it easier for the user to choose which route he wants to draw with our application, we had an idea of listing all the possible routes so the user can see them, and then choose one. But, since the data has routes from the whole Austria, that was not really practical. After merging the data, we got the table with all the routes possible, and there were more than thousands of them. Since the listing happens in the Terminal, it was not a practical way to see them. But that was not the only problem with this approach. With this way, the user would not be able to see all the stops that the train goes through, and maybe he needs to see the trip between one stop which is not the first or final destination. There was not really an easy way to do this. In the end, we decided to list all the stops from the dataset. Then the user can choose any two stops, which is the best idea since he wants to see the trips between those two stops. If there are no trips available, he would get a message saying that, and if there are trips available he would get the CSV ready for drawing.

8.3 Non-linear stops

The time taken to travel between stations is used as the distance metric for separating stations on the x-axis. However, problems arise when stations are not visited in the same order by all trains, or trains take a different amount of to travel between stations. This can create zig-zag lines(Figure 8.1) where

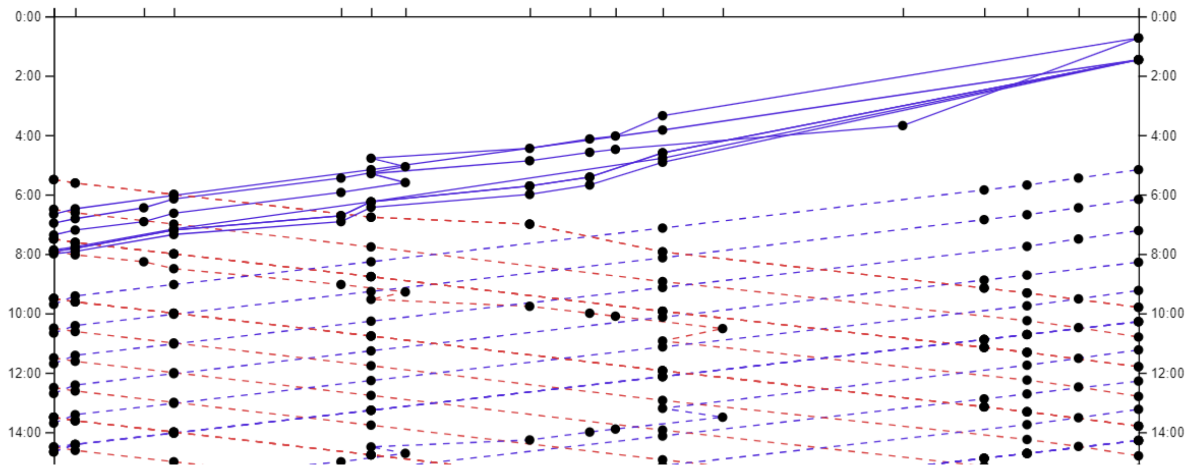


Figure 8.1: Zig-zag lines caused by non-linear train stops. [Screenshot made by Elliot Koh using String Charter.]

trains appear to be going backward. Several mitigating workarounds were tested to varying degrees of success but unfortunately, no good solution to this problem was found.

Chapter 9

Conclusion

The Ibry Train Schedule has a long and storied history and it is a worthwhile endeavour to recreate it for use on the Austrian railway system. To do so, tools such as Python, TypeScript, D3 and Electron were used. A script was created to pull and format data from the ÖBB website and used as the basis of this project. The script was also flexible in that it did not need to be limited to only Austrian railway system but it could show any railway system that uses standardized GTFS way of storing the datasets. Additional features such as filtering, colour customization, tooltips, and an SVG export function were added. The design of the application went through several mockup iterations before the final version of the application.

Bibliography

Bostock, Mike [2020]. *D3.js - Data-Driven Documents*. https://d3js.org/. Jun 2020 (cited on page 5).

Foundation, OpenJS [2020]. *Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS*. https://www.electronjs.org/. Jun 2020 (cited on page 5).

Google [2020]. *GTFS Static Overview*. https://developers.google.com/transit/gtfs. Jun 2020 (cited on page 7).

Microsoft [2020]. *TypeScript - JavaScript that scales.* https://www.typescriptlang.org/. Jun 2020 (cited on page 5).

MockFlow [2020]. *MockFlow*. https://www.mockflow.com. Jun 2020 (cited on page 17).

Python [2020]. *Welcome to Python.org*. https://www.python.org/. Jun 2020 (cited on page 5).

Rendgen, Sandra [2019]. *History: From Paris with Love (ca. 1845)*. https://sandrarendgen.wordpress.com/2019/03/15/d trails-from-paris-with-love/. 15 Mar 2019 (cited on page 3).