Advanced Data Visualization
Project 2
Electric Bus Deployment Visualization

Luis Rodriguez-Garcia (u1267015)
Mehdi Ganjkhani (u1268148)

Due date: October 25th, 2021

1 Introduction

The objective of this project is to create an interactive visualization prototype of electric bus deployment data using D3.js. D3.js is a JavaScript library for producing dynamic, interactive data visualizations in web browsers. The clients of this projects are domain scientists (Cathy Liu, Jianli Chen, Yirong Zhou from the Department of Civil & Environmental Engineering) who work on smart city design. The input data for this project is created based on battery-electric buses (BEB) optimization deployment considering cost and environmental equity for disadvantaged population. The output data of the optimization problem as:

- Locations and number of both in-depot and on-route charging stations
- Number of buses to be replaced
- The exact buses to be replaced

The Utah Transit Authority (UTA) is selected as case study for this project and three different plans with various budgets ($25M for plan 1, $60M for plan 2, and $120M for plan 3) are taken into account.

We used the result of the optimization problem as input data in order to design a web page and enable user interaction using D3.js as the main objective of the project. This report briefly explains the steps we passed in order to overcome the project and address the requirements. The remainder of the report is as the following: Section 2 explains the data wrangling process. In this section, we explain all the process we have done on data in order to use it in D3.js and the web page. Section 3 elaborates the components of the visualization and an explanation for each component. Section 4 provides information related to the implementation and the functions and libraries used.

2 Data wrangling

This section explains all the pre-processing and data wrangling we have done for this project to prepare the data in an appropriate format, filter and extract only the useful data and form it in an appropriate format to be able to read it on the webpage using D3.js. The provided data for the project by the client is as follows:

- Network Data
  - Bus stops — shapefiles
  - Bus routes — shapefiles
  - Runcut file — CSV
- Deployment Plans
  - Solution file for each plan — txt

    - UTA Runcut PotentialStop — xlsx

- General information about the battery capacity and mileage and explanation

2.1 Shapefile data wrangling

The information of routes is stored in shapefiles, containing information of all bus routes and bus stops administered by UTA. To represent the routes in our implementation, initial and final stops of each routes need to be identified, as well as the path connecting both ends. In order to extract the required information, the next steps are followed:

2.1.1 Initial shapefile visualization in GIS software

The shapefiles are loaded using QGIS, a free tool for GIS data visualization and manipulation. In this step, we verify that the routes are loading properly, that the bus stops are located on the routes, and that the coordinates of all points match with Salt Lake City and surrounding areas.

2.1.2 Exporting routes

Drawing all the routes and bus stops in the developed interface is not necessary and is computationally expensive. Instead, a limited set of bus routes is defined according to the information extracted from the text files, identifying the buses that are replaced by BEBs and the routes associated with these buses. In total, 57 routes are related to buses converted into BEBs for plans 1 to 3. These 57 routes are selected and exported as a KML file.

2.1.3 Route redrawing

Even though it was originally intended to extract the coordinates of each route segments from the KML file, it turned out that the segments of each routes were not sorted in such a way that connects the initial and final stops are sequentially connected on each route. In that case, the routes had to be redrawn to create a sorted set of segments connecting each end, so that locating the bus in the route was possible. To do so, the extracted KML file is loaded in Google Earth. For each route, the initial stop is located and a reduced set of segments is drawn based on the original route sketch, until the final stop is reached. The redrawn routes are then exported from Google Earth as a KML file.

2.1.4 Coordinate extraction

A Python code is created to read the coordinates of each segment on each route from the KML file, which sorts the information within tags. The file is read line by line. Once a new route is detected, the line number and line name are stored, as well as the latitude and longitude of each end. For each segment, the following information is stored:

- Line code (referred as `LineAbbr` in the JSON file)
- Line name (referred as `LineName` in the JSON file)
- Latitude of the initial segment point (referred as `latA` in the JSON file)
- Longitude of the initial segment point (referred as `longA` in the JSON file)
- Latitude of the final segment point (referred as `latB` in the JSON file)
- Longitude of the final segment point (referred as `longB` in the JSON file)
- Name of station (if it is the initial or final segment, referred as `station` in the JSON file)
- Cumulative distance from the initial segment to the final point of the segment (referred as `CummulativeDistance` in the JSON file)

2.1.5 Exporting file

The organized file is stored as a JSON file using nested dictionaries, where the most external key corresponds to the route code, followed by the segment number and then the before mentioned information. The extracted file can be found in the dataset folder as `dataset/coordinatesFinal3.json`.

2.1.6 Bus stop coordinate extraction

As of the bus stops, a simpler procedure is executed. The bus stops are exported using QGIS as a KML file. Then using a Python code, each coordinate is extracted. These are stored according to their name, which is used as a key, and each item containing latitude and longitude. The extracted file can be found in the dataset folder as `dataset/stopsFinal.json`.

2.2 Deployment Plan data wrangling

The information of each plan is stored in text format, containing the information of the charging sequence, the accumulated mileage of each bus at each sequence, the charging stations and the BEBs. In order to extract the required data for each deployment plan we used these files in each deployment plan alongside with the Runcut file. A Python code, namely *"Data_preprocessing_final.py"* is used for the data wrangling.

The input files of the code are:

- `1. Network Data/3. UTA Runcut File Aug2016.xlsx`
- `2. Deployment Plans/1. Solutions/p20.txt`
- `2. Deployment Plans/1. Solutions/p60.txt`
- `2. Deployment Plans/1. Solutions/p180.txt`
- `routeMileage.xlsx`
- `2. Deployment Plans/2. UTA_Runcut_Potential_Stop.xlsx`

The file `routeMileage.xlsx` contains the mileage of each route extracted in previous section for 57 routes which is used to estimate the distance of the bus from the last station based on time when the BEBs are on route. Also this file is used to estimate the accumulated mileage for the BEBs whose daily traveling distance is less than 62 miles. (Since this data is not provided in each plan)

The file `UTA_Runcut_Potential_Stop.xlsx` contains all the stops regardless of being charging station or regular station. This file is used to provide a list of regular stops for each plan.

The output files of the code are:

- dataset/final_data.json
- dataset/final_cs.json
- dataset/non_cs_data.json

Each output is explained in the following:

2.2.1 final_data

The `final_data` file is created as a dictionary in Python with different keys and values. The file provides required information for each BEB to show in the visualization based on each plan, time input and each BEB. The structure of the file for each plan is as: {*"Plan #":{"BEB": {"time": {"SoC":, "Mileage", "Route_number": , "Location":, "Distance_from_last_station":, "Route_length":, "route_percentage":, "Charging_status":*}}}}.

The required information is extracted from each plan and the UTA Runcut file is used to extract the time dependant information. More detailed information about the implementation can be found in the `Data_preprocessing_final.py` file.

2.2.2 final_cs

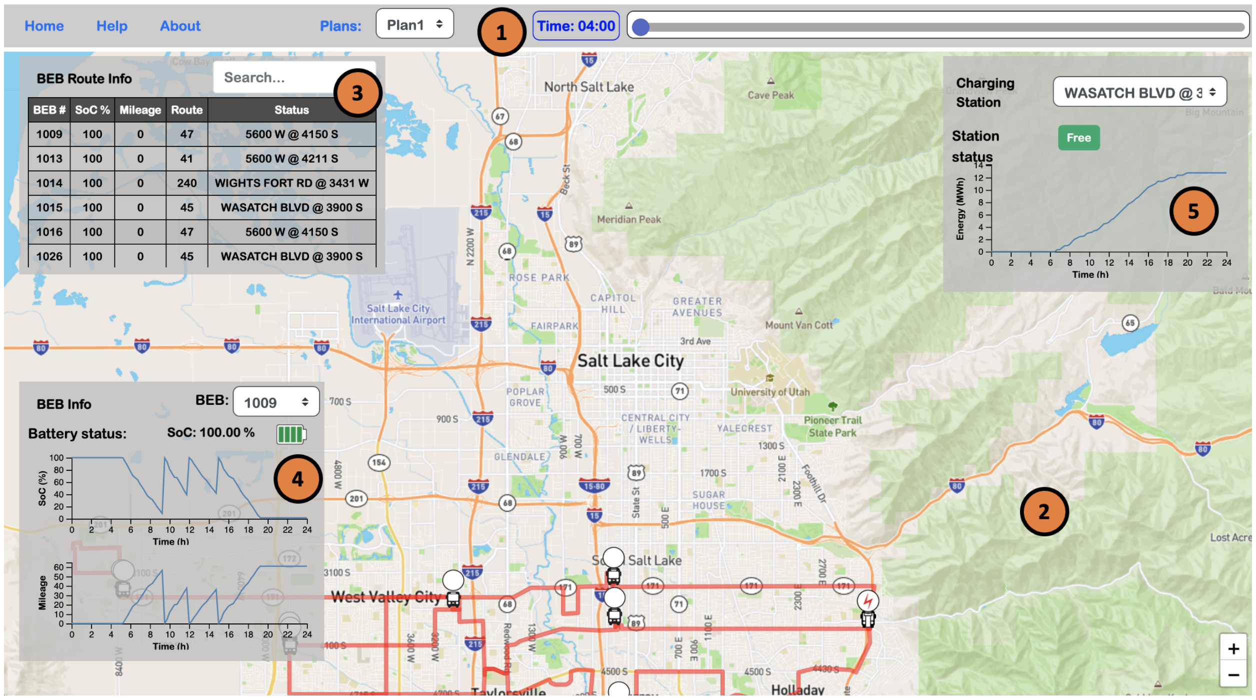
The `final_cs` file is created as a dictionary in Python to provide required information for each BEB to show in the visualization based on each plan, time input and each charging station. The structure of the file for each plan is as: {*"Plan #":{"Charging station #": {"time": {"charging_status":, "Bus":, "seq":, "energy":}}}*}. The key `"charging_status"` specifies if the charging station is free or in-use. Worth mentioning that the key `"energy"` specifies the accumulated energy usage for each day. Also instead of adding 400 kWh for each time the charging station is used, the previous charging status is taken into account to obtain more accurate result. The required information is extracted from each plan and the UTA Runcut file is used to extract the time dependant information. There was an inconsistency in the `"p60.txt"` and `"p180.txt"` showing that the specified charging stations ("Y##") do not match with the stations that the BEBs are charged based on the sequences. We considered the charging stations based on the sequences since those were more comprehensive. More detailed information about the implementation can be found in the *Data_preprocessing_final.py* file.

2.2.3 non_cs_data

The *non_cs_data* file is created as a dictionary in Python to provide the list of non-charging station stops in order to illustrate in the map. In order to do so, the list of all the station is imported for each plan and the chargin stations are excluded from the list. More detailed information about the implementation can be found in the *Data_preprocessing_final.py* file.

3 Visualization

Alongside with the data wrangling, the visualization is designed and implemented. The visualization result is shown in Fig. 1. The visualization panel contains five major components: the top navigation bar, the map, the BEB route information, BEB information and charging station. The interaction and the visualized information for each component are explained in the following:



BEB #	SoC %	Mileage	Route	Status
1009	100	0	47	5600 W @ 4150 S
1013	100	0	41	5600 W @ 4211 S
1014	100	0	240	WIGHTS FORT RD @ 3431 W
1015	100	0	45	WASATCH BLVD @ 3900 S
1016	100	0	47	5600 W @ 4150 S
1026	100	0	45	WASATCH BLVD @ 3900 S

Figure 1: An overview of the visualization panel

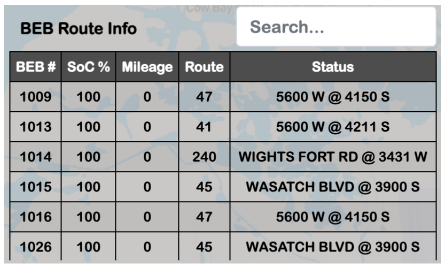
3.1 Top Navigation Bar

The top navigation bar provides navigation between the Home, Help and About pages. The Home page is the main page shown here. In Help page we provided a legend for each icon we have used in the visualization in addition to an explanation about the interactions and sections. The About page summarizes the project and provides a brief introduction of the partners. In Plans selector, the user can select among the plans and the information of the page is automatically updated accordingly for 04:00. Using the time, the user can select a time from 04:00 to 24:00 with 30 minutes time interval. All the other parts of the interface are updated accordingly.

3.2 Map

The map component displays the status on electric buses, routes, and charging stations on an interactive map where the user can easily visualize information. The routes displayed in the map are those where electric buses are assigned according to the plan. In addition, by using the time selected in the top bar, the location of buses and status of the charging stations is updated. The user can visualize information about the routes by clicking on the route, which opens a pop-up with the route name. The user can also visualize information of the buses by clicking in their respective icons, which shows if the bus is on route or at a charging station, the state of charge, the current route, or current charging station (if charging). Charging and non-charging stations are also displayed in the map. By clicking on the stations, users can see the name of the stations. If the user clicks a charging station icon, a pop-up includes the total energy consumed by the charging station, from midnight till the selected time. The user can interact with the map by zooming (among maximum and minimum zoom limits that do not distort the visualization) and panning through the map to explore details.

3.3 BEB Route Information

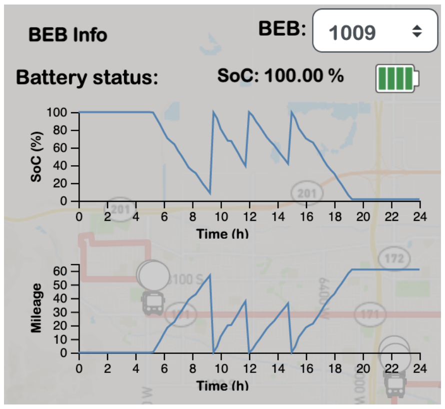

BEB Route Info

Search...

BEB #	SoC %	Mileage	Route	Status
1009	100	0	47	5600 W @ 4150 S
1013	100	0	41	5600 W @ 4211 S
1014	100	0	240	WIGHTS FORT RD @ 3431 W
1015	100	0	45	WASATCH BLVD @ 3900 S
1016	100	0	47	5600 W @ 4150 S
1026	100	0	45	WASATCH BLVD @ 3900 S

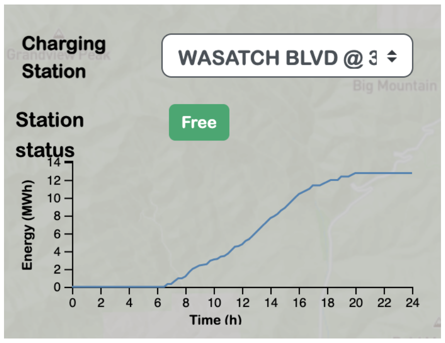
This component illustrates the information of all the BEBs in each plan. The table contains a list of the BEBs. The second column provides state of charge (SOC) for each BEB based on the selected time, Mileage shows the accumulated mileage of the BEB after the charged last time. Route provide the information about the number of the route that the BEB is in or is going to be in in the specified time. Finally, Status of each BEB determines of the BEB is on the route or in an station. In former case, the status would be written as ”On route”, in the latter case, the name of the station is written in the column. A search box is provided for the user to interact with the table and filter it based on the desired input.

3.4 BEB Information



The BEB Information panel shows specific information of the buses of each plan. The user can select the specific bus they want to explore by clicking on the drop-down list. Once a bus is selected, the panel is going to display the state of charge as a percentage. In addition, an icon is displayed indicating the charging status, showing if the bus is fully charged (green icon with four lines), mostly charged (green icon with three lines), mid charged (yellow icon with two lines), or low charged (red icon with a one line). In addition to the battery status, this panel displays the state of charge and mileage as a function of time. The charts show the cumulative state of charge and mileage depending on the selected bus and time, starting from hour 0 (00:00) to their selected time.

3.5 Charging Station information



This component depicts individual information for each charging station. The charging station can be selected by the user in the top right drop-down menu. The drop-down list is updated for each plan. Once the charging station is selected, the status of the charging station and the energy figure is updated accordingly. The station status illustrates if the station is "free" or "in-use" in the given time. If the charging station is in-use, the number of the bus being charged is appeared. Finally, the consumed energy in the given charging station is drawn from 12:00 to the given time. Using this information, the user can analyze and monitor the energy consumption of each charging station during the day.

4 Implementation

The implementation of the interface requires an interaction between HTML for creating the web page, CSS for styling, Javascript to create and control the interactive elements on the web page. Several Javascript libraries were used to set functionalities, and their use is briefly explained below.

4.1 Top Navigation Bar

The top navigation bar is implemented using `booststrap`[1], an open-source CSS framework. The navigation bar is created by defining a `div` in the beginning of the body. Also the time slider and the plan selection drop-down are implemented in this `div`.

4.2 Map

The map is implemented using `Leaflet`[2], an open-source Javascript library to create interactive maps. The map is created by defining a `div` with a specific height. Then, the `div` is called to place the map. Leaflet creates the map and adds a layer using a Mapbox token. The map is centered in a specified coordinate (in this case, in the middle of Salt Lake City UT). Other elements on the map can be easily added with Leaflet built-in methods. Routes are drawn using `polyline`, where each of the plan routes is drawn using the sequence of coordinates extracted in Section 2.1. The buses and charging stations are located using `marker`, which has the advantage to be customized to any design and attach pop-ups with information related to the buses or the charging stations.

Icons on the map interact with the zoom, scaling properly when the user zooms in and out. This is achieved using built-in functions that interact with events occurring on the web page. Specifically for the icons, the zoom is adjusted with a `zoomend` event, which scales the zoom at the end of each zoom step according the its value.

Buses are located by scanning the list of buses in the selected plan and time. For each bus on the list, it is determined if the bus is either on route or located at a station (which can be charging or non-charging station). If the bus is at a station, a bus icon is located at the station coordinate. If the bus is on route, the icon is located at a estimated location based on the distance driven from the last station (which is stored in the route coordinate data). In addition to the icons, the routes are highlighted when a bus is selected from the drop-down list in panel 4.

4.3 BEB Route Information

A `div` is created for this section and a table is located in the `div` with predefined style in CSS. In order to enable "search" in the table, a script is implemented in the HTML. The table is updated based on the different inputs from the user including plan and time. The table is initialized based on Plan1 and time 4 AM using D3.js. The information in the table is updated using the functions written in the script and handled by D3.js. The table is updated once the plan or the time is changed.

4.4 BEB Information and Charging Station

A `div` is created for the BEB information section to illustrate the information for each BEB as explained earlier. The BEB drop-down list is updated using an implemented D3.js function based on each plan. Another `div` is created for the charging station section and the charging station drop-down list is updated using an implemented D3.js function based on each plan. All the SVGs and, battery icon, SoC and station status are controlled and updated using the built-in functions in a D3.js script. The information in these sections are initialized for Plan1, time 4:00 AM and initial BEB and charging station. Then the information in these sections are updated and shown based on the user's interaction. The implementation can be found in the "Main.html".

5 Running the interface

The tool needs to be run using a local server. Node.JS is recommended

1. Install Node.JS (https://nodejs.org/en/)

2. Using the terminal/command prompt, run the command: `npm install --global http-server`

[1]https://getbootstrap.com/
[2]https://leafletjs.com/

3. Once installed, locate the project folder in the terminal, then run: http-server
4. Open the server IP address into the web browser to open the index, then select Main.html

Detailed step by step is shown in the video explanation, which can be found at https://youtu.be/URBzca6angw (uploaded as Unlisted and it can only be accessed with this link)