Real-Time Seattle Bus Statistics and Prediction

Michael Makris - 4/28/2016

Project Repository

https://github.com/MPMakris/BUSES

Question

This project's question is in two parts:

- Is it possible to provide accurate predictions of bus arrivals and trip time based on past bus data?
- Is it possible predict which bus a user should take in real time, when given a choice, to best improve trip experience?

Introduction and Problem

Seattle's bus network is known as the birthplace of One Bus Away (OBA), an open server- and application-package for cities to publish their real-time bus information for riders to use. It can be easily demonstrated, however, that the application showing bus riders when their vehicle will arrive can be imperfect, with predicted arrival times changing minute by minute, and some busses not showing any real-time data at all.

What is missing from the application is an informative and predictive resource for riders to see past bus performance of their intended rides and predict that day's arrival and trip times. This project aims to produce an informative and predictive resource for bus riders to better understand and see their bus's performance in real time.

To accomplish this, the project aims has several milestones:

1. Collect and store real-time data on Seattle bus positions over a period of days, as well as transit agencies' predictions for arrival over that same time.
2. Clean, collate, and parse the data into manageable subsets, including by agency, by route, and by stop.
3. Create a database of the expected arrivals of each bus for each day, to compare against when busses actually arrived. This data will come from the transit agencies' published schedules.
4. Analyze the data to calculate bus locations over time, as well as bus performance as compared to expected performance in published schedules.
5. Use past performance data to create models of future bus performance, and predict bus performance in real-time based on current conditions.

Obtaining the Data Set

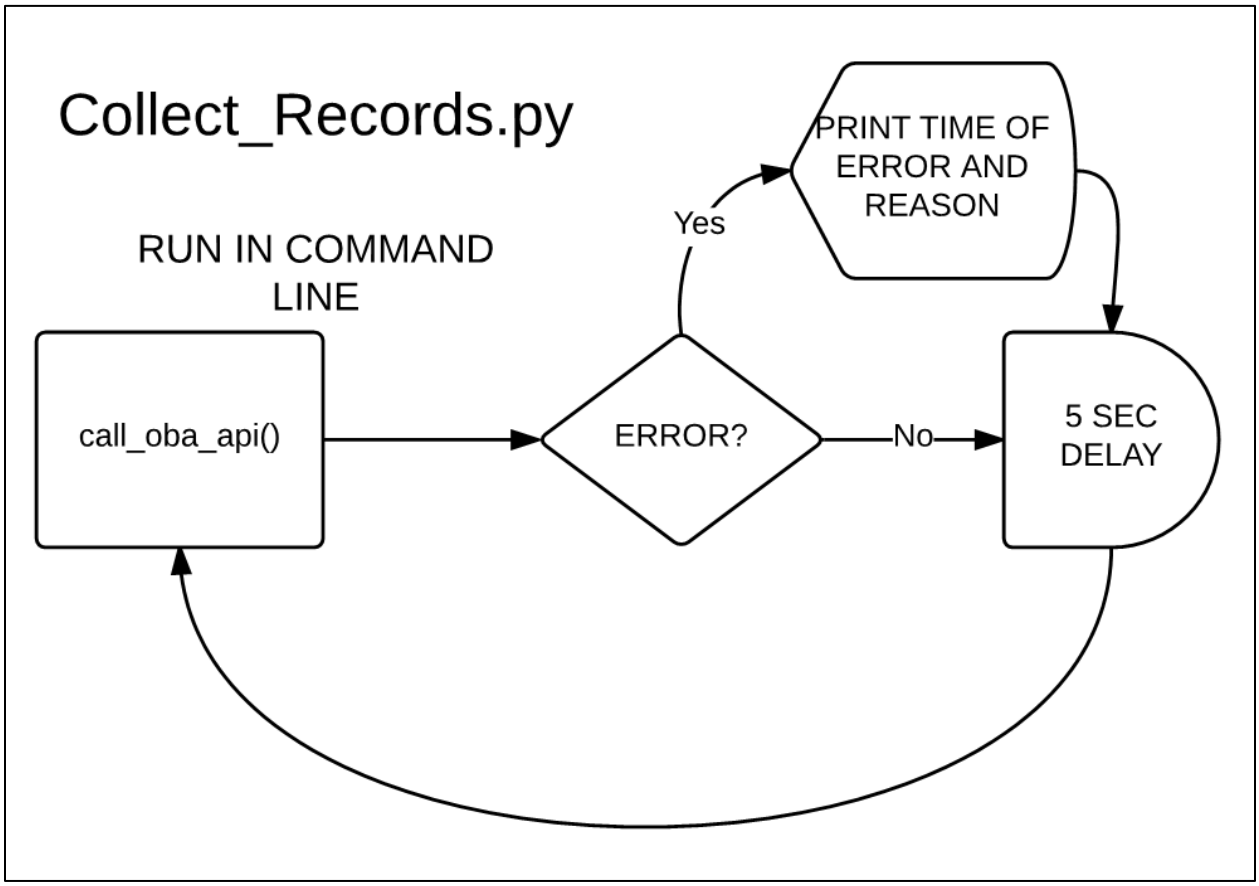
Seattle transit agencies do not keep historical records of their real-time bus data in any format accessible to third parties. Instead, they allow interested parties access to the OBA API, which publishes real-time data on command. The available data for each of the participating agencies is as follows (with given API command):

- Bus Positions (Vehicle-Positions-For-Agency)
- Trip Updates (Trip-Updates-For-Agency)
- Agency Alerts (Alerts-For-Agency)

To collect a large enough data set to analyze historical trends, bus positions and trip updates needed to be collected over a designated period of time. A python script was created that would automate the collection on a timer. *Collect.py*[1] runs on a defined loop and pulls the above data and saves it to a unique .CSV file. Saving the data is not sufficient, however. The raw data pulled from the agency servers is in a format not readily converted to tables.

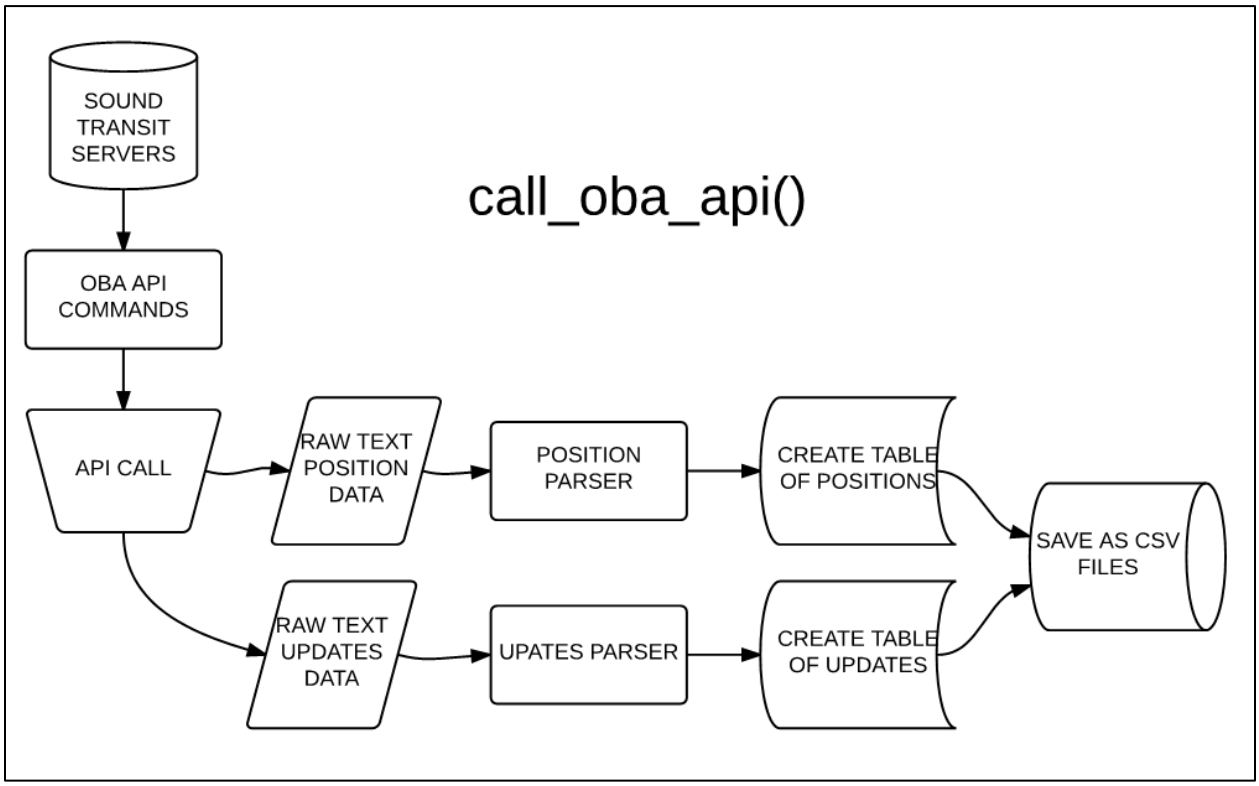
Collect.py first parses the data. The raw data is organized in a format similar but crucially different from .XML. Normal JSON and XML interpreted failed to read the data. Instead, a routine was built custom for each data type (position, update, alert) that organizes the data into lists (including Agency Id, Trip Id, Vehicle Id, Vehicle Positions, etc.), and then each list is inserted into a column of a Pandas DataFrame. This process is called an OBA Call. Once the DataFrame is complete for a particular API call, the data is written to a CSV file, the python timer resets, and then after the designated time, another call is made to get data from the servers and the process repeats. See Figures 1 and 2 below.

FIGURE 1



[1] https://github.com/MPMakris/BUSES/blob/master/Python_Scripts/Collect_Records.py

FIGURE 2



With Collect.py written, it was set to run for 8 days on a timer of 5 seconds, or approximately 140,000 calls. However, as in ran over the course of several days, errors would sometimes cause it to fail, including slow or no internet access, as well as slow response from the agency servers. These problems were intermittent, however, and did not prevent a robust set of over 100,000 calls from being made over 8 days.

Collec.py only calls from one agency at a time. For this project, King Country Metro was chosen as the sole agency to record, as it contains the vast majority of Seattle's buses as well as the most immediate impact on users in Seattle. Collect.py could easily be updated to collect for other agencies just be changing the agency_ID variable.

Cleaning and Collating the Data Set

While data collection proceeded, cleaning of a smaller sample of data began. A process was developed to take the individual CSV files for each OBA Call, and merge them into a master file for the agency under consideration. A Jupyter/ipython notebook called *Create_INSTANCES*[2] was created to collate all the positional data into a master list of INSTANCE. INSTANCES are the term used in this project for each line of historical data. That is, each row of data showing a particular bus at a particular time is an INSTANCE. With near 100,000 position CSV files, and roughly 260 buses contained in each, there were about 26,000,000 INSTANCES created for this project to work with.

Those raw INSTANCES still required cleaning and additional data, however. First, the OBA API reports all bus positions it has info for at all times, even if the bus is not in service. Therefore, many INSTANCES do not contain route or trip information, making them useless. These INSTANCES were removed from the master file of INSTANCES. Second, the positional data does not include any information on bus stops, routes, when the data was

[2] https://github.com/MPMakris/BUSES/blob/master/Notebooks/Create%20INSTANCES.ipynb

called, or anticipated times of arrival. A series of joins had to be performed to properly identify routes, and timestamps assign them to the proper INSTANCES.

This notebook also then breaks the INSTANCES into affiliate routes, saving a single CSV file for each route on King County Metro's schedule. This will allow later analysis work to access only the data on the route in question when needed, rather than having to access the entire 26,000,000 INSTANCES each time data is required.

In addition to Create_INSTANCES, another notebook was created, *Create_EXPECTS*[3], to search through all the scheduled stops in the published bus schedule and identify when each bus should arrive where. This list of EXPECTS will allow further analysis to be done to determine if busses arrived on time when they were expected.

The process of creating EXPECTS was particularly difficult, since OBA servers do not organize their schedules in a single file. Instead, they are organized into nine different TXT files that are relational[4]. Create_EXPECTS must collate information from many of these TXT files to produce the full list of EXPECTS for the agency.

Analysis

The project is currently in the analysis phase. Determining where buses are at a particular point in time is relatively simple using a library called geopy[5]. However, comparing which busses should be measured against which stops at which times is far more complicated.

To work through how to determine which buses should be compared to which known bus stops at which time, another notebook was created, *Analyze Route Data.ipynb*[6]. This notebook is showing success in filtering out busses not expected at a certain stop based on which route is under consideration. Further work will loop through all routes and all stops and create pairs of pre-organized data showing when busses arrived at each stop. Then, using the known EXPECTS for each stop, an analysis can reveal how those buses performed.

Current Data Set

The total data sets for the project far exceed Git Hub's free limits for data storage. However, using the scripts and notebooks detailed on the project's Git Hub page, it is possible to recreate a new set of data by called from the OBA server.

[3] https://github.com/MPMakris/BUSES/blob/master/Notebooks/Create%20EXPECTs.ipynb
[4] See Sound Transit: http://m.soundtransit.org/Developer-resources/Data-downloads
[5] https://pypi.python.org/pypi/geopy
[6] https://github.com/MPMakris/BUSES/blob/master/Notebooks/Analyze%20Route%20Data.ipynb