Samantha Karam, Sarah Medeiros
Data Mechanics

Optimizing Boston Public School Bus Stop Locations

Transportation costs for Boston Public Schools currently account for 11% of the district's budget. Considering this percentage is larger than the corresponding value for larger cities like New York, Boston should be able to optimize their use of buses to cut costs so that less money is wasted on transportation and more money is used to better the education that students' receive. In this project, we have focused on a subset of the bus problem. Our goal was to optimize the location of corner bus stops so that buses could pick up the maximum number of "corner stop" students (students that did not need to be picked up at their doorstep) at the least number of stops while also taking into account the distance a child is allowed to walk to reach their stop. This is the first step in improving Boston's transportation system for their public schools.

Two of the datasets that we chose to work with are the students-simulated.geoson dataset and the school-real.geojson dataset. Both of these were provided by the city of Boston for a competition to solve the bus stop optimization problem. The students-simulated.geojson dataset included generated student data including their school, address, max walking distance allowed, ect. Actual student data was not publicly released to keep their personal information safe and private. The school-real.geojson dataset is a collection of all of the Boston Public Schools including their start time, end time, and address (as well as their corresponding latitude, longitude coordinates). The last dataset we used was the Boston, MA geojson (OSM2PGSQL) dataset which we retrieved from the following site:

https://mapzen.com/data/metro-extracts/metro/boston_massachusetts/.

This dataset gives us Boston's location data in the form of geojson which we use to find the all of the street corners within the city.

We begun tackling the problem by storing the information from the students-simulated.geojson file into a database. We decided simplify the problem and only took into account their location, school, and their status as corner to corner students. However, we did store all of the information about the students - not just the fields we were working with. This is

because it is important for the final solution to take into account things like getting the students to school on time, the maximum amount of students riding the bus, and more. Solving these issues parallelly has proven to be too difficult, however, the data is available so we can later further our solution.

We employed the k-means algorithm to choose preliminary bus stop locations. The actual bus stops were required to be corners, but we fit this constraint in a later step. First we grouped the students by school and ran k-means on these groups. As these are public schools, the students from a single school would be in some set vicinity since school districts are separated by location. Additionally, this allows for logical bus stop assignment as all of the kids assigned to a single bus stop will be going to the same school. The separation by school was particularly helpful for our visualization. Given a new child, we only needed to recompute the stops for that child's school. We tried multiple k's to try to minimize the amount of stops while also trying to keeping students from having to walk more than .5 of a mile. In our visualization, we chose to assume that any student added to the database would be able to walk no more than .5 of a mile. In reality, some students, depending on grade and neighborhood safety, may only be allowed to walk .4 miles, .3 miles, or .2 miles.

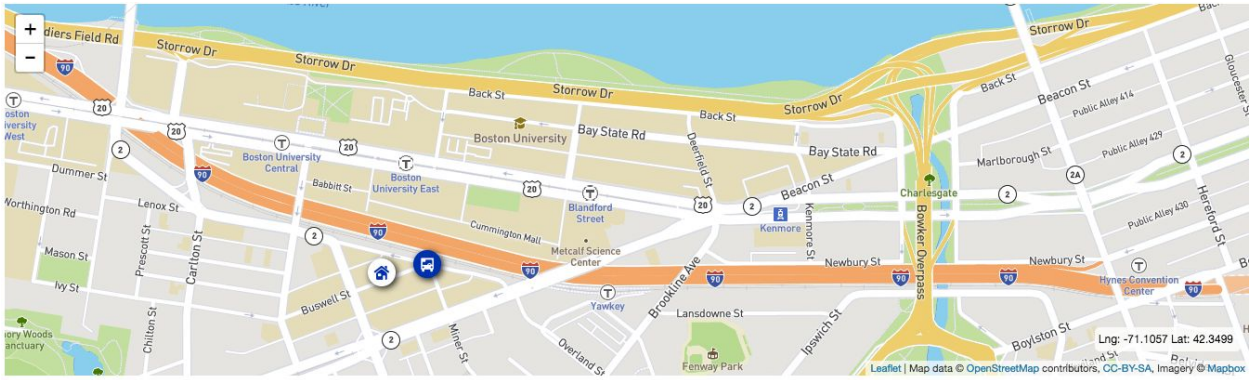
After finding the best places for the stops, we had to deal with another constraint problem. First off, our k-means algorithm could have found an off road point, so we would have to find the closest road. Additionally, the city of Boston requires that children must be picked up at a corner unless they have a wheelchair or other special needs that requires a pickup at their doorstep. Therefore, we mapped our results from k-means to the closest possible street corner. We used the city of boston geojson dataset to get all of the intersections in Boston. The challenge of using this dataset was the sheer size proved difficult to efficiently traverse through. To speed up runtime, we followed the theory behind r-trees. The goal of r-trees is to group nearby items and represent that group by a minimum bounding rectangle. This tactic only requires a search of the points within the minimum bounding rectangle (rather than iterating through all of the possible points). Therefore, to combat the issue of size, we split up the dataset into searchable subsets for each school so that we could minimize the amount of intersections that we had to

check for possible bus stop placement. This significantly decreased runtime however, the runtime is still significantly long.

Once we found feasible corner bus stops, we checked to see if the students could still walk to those stops as they now have shifted from their original placements. We stored whether the student was in reach of the stop or not. That way in the future, we could re-run k-means for only those stops using a larger k until all students could reach their respective stops within their assigned walking distances.

We additionally chose to also make a webservice to allow the input of new students into the database and to return their bus stop. In order to calculate this information, we ask the user for their home address and their school (which can be chosen from a dropdown list). To populate the dropdown list, we modified the school-real file to retrieve only the school names.We then used this list to fill the dropdown in the html. Originally, we were computing the latitude and longitude values in the javascript of our html page. To do this we made a request to the Google Maps API to convert the address to latitude and longitude. However, we ran into issues with this method later when we needed to retrieve the new bus stop coordinates from our python functions. To combat this issue we changed our code so that we were retrieving the address and school name in our Python file. This fix made calculating the coordinates of the address even easier because we no longer needed an API key and we could just import Geocoder.

After we obtained the coordinates and the school that the student will attend, we run k-means on the set of students that go to the new student's school (including the new student). Then we map the new stops found from k-means to their closest corners. We then used Leaflet to display on a map the new stop to the user and to show the end user the address they inputted. In order to differentiate between those two points, we used MapShakers MapKey Icons to change the image and color of the Leaflet markers. In order to run the visualization web service yourself, please follow the directions at the bottom of the page.



Enter your address and the current school that you attend to be added to our student database, so that we can find your bus stop!

Your Address:

Your School:

Academy Of Pacific Rim

Submit

This problem is extremely complex and our solution is far from complete. First off, optimizing the search for corners could greatly improve our solution's implementation. Currently, this is the main factor in the long run-time. Also, we are not fitting each child's walking distance constraint. Currently, we only run k-means once with a k we settled on after some brief experimentation. Then at the end, we return whether all of the children for each school were able to reach their stops or not. However, we do not go back and run k-means with a larger k(as this would extend the already lengthy run time) although this is a necessary step. Finally we ignored some important constraints in their entirety such as school start time, school end time, bus start locations, and bus capacity. To fully solve this problem, we need to be able not only minimize the amount of stops, but also the number of buses on the road. Additionally, we need to make sure the children can get to school on time and within one hour of getting on the bus(another constraint imposed by the city).

Another interesting improvement that could be added to our web service would be for the new student's data to be shown alongside students that attend that same school and their corresponding bus stops. This addition would help us visualize how adding a new student adjusts the assignment of bus stops. Adding students that are far away from the school and other students would likely cause the addition of a new stop close to them while it is more likely that closer students are assigned to a preexisting bus stop.

References

http://datamechanics.io/

http://cs-people.bu.edu/lapets/591/

https://mapzen.com/data/metro-extracts/metro/boston_massachusetts/

https://github.com/Data-Mechanics/geoleaflet

http://leafletjs.com/

http://bostonpublicschools.org/transportationchallenge

https://github.com/MrMufflon/Leaflet.Coordinates

https://github.com/mapshakers/leaflet-mapkey-icon

http://flask.pocoo.org/docs/0.12/quickstart/

How to run our visualizations:

Run Mongo:

    Mongod --auth --dbpath /data/db

Run these python files first in order:

    python getStudents.py

Make sure you are ready to use Flask by following the quick start guide to setup your machine.

http://flask.pocoo.org/docs/0.12/quickstart/

Run the webservice:

    python webservice.py or flask run

After the service is running, go to localhost:5000/home.

If you go to localhost:5000 you will be redirected.