Can Traffic Sensor Data Detect Vehicle Cruising?

Brett Bejcek
UW DSSG Fellow
The Ohio State University

Anamol Pundle
UW DSSG Fellow
University of Washington

Orysya Stus
UW DSSG Fellow
UC San Diego

Michael Vlah
UW DSSG Fellow
University of Washington

Vaughn Iverson
Data Scientist
eScience Institute

Valentina Staneva
Data Scientist
eScience Institute

Steve Barham
Project Lead
City of Seattle

July 28, 2017

ABSTRACT

Vehicle cruising (individuals looking for parking and for-hire vehicles operating without a passenger) is a major contributor to traffic congestion in downtown Seattle. Still, the magnitude and location of vehicle cruising is poorly understood. To get a better understanding of where vehicles cruise, we propose a framework for using traffic sensor data. We generate most likely paths traversed through filtering out unrealistic behavior and incorporating routing. We break up individual trips via segmentation in terms of time and method of transportation. To guide future machine learning algorithms we visually label paths which appear to be cruising and propose a list of attributes to train the cruising classification algorithms. Ultimately, we create a preliminary heatmap of downtown Seattle that can be used to visualize the relative levels of cruising, once they are obtained from the machine learning algorithms.

OVERVIEW

Cruising has a significant impact on congestion and travel time reliability. Within the scope of our work, we define cruising vehicles as those that have already arrived at their destination and are driving around looking for a place to park as well as taxis, for-hire vehicles, and transportation network companies (TNCs) that are queued in traffic waiting to be hailed or to pick up another customer.

As part of the Data Science for Social Good Program at the University of Washington, we are developing algorithms to quantify aggregated levels of vehicle traffic cruising by applying data science techniques to analyze a sample of anonymous travel sensor data.

We have plans to differentiate between the aggregated footprint of vehicles trying to find on-street parking and the amount due to trip deadheading (vehicles for-hire that are driving without a passenger). This research has the potential to help transportation agencies, technology companies, and car companies predict the availability of parking and more accurately direct travelers with online, mobile, and connected tools, thereby reducing congestion, emissions, and fuel costs.

The end goals of this project are to synthesize a sample of anonymous sensor data into a heat map depicting the relative prevalence of cruising and to propose measurement standards for cruising activity, such as a cruising index. This approach could be incorporated into real time applications and would be a potential candidate for use in a linked data repository with strong governance, such as the University of Washington Transportation Data Collaborative.

BACKGROUND

The City of Seattle seeks to improve travel reliability, optimize the use of the right-of-way, improve the parking experience, reduce emissions, and lower transportation costs. There are two forces that are thought to be counterproductive to these goals:

1. Cars searching for parking spots after they have already arrived at their destination. There have been studies[1] that suggest that as much as 30 percent of traffic in congested areas could be due to parking cruising. A new study released by Inrix suggests Seattle parkers spend 58 hours per year searching for parking and ranks Seattle as number five in the U.S. worst cities for parking[2].
2. Vehicles for-hire (taxis, for-hire vehicles and app-based Transportation Network Companies) queuing in motion or waiting to be hailed. For every for-hire trip, there is inevitably a measurable amount of travel without a passenger, or deadheading. The share of vehicles for-hire is increasing at a faster rate than most other transportation modes and has profound implications for the city's transportation network if current trends continue.

In both cases, the City of Seattle has very little data regarding vehicle traffic cruising and has difficulty measuring the impact on Seattle streets. Better data would support informed transportation policies, use of the right-of-way, and infrastructure investments.

METHODOLOGY

ANONYMIZED DATA

This project is exploring the use of traffic analysis sensors. The sensors have been placed in downtown Seattle to calculate travel times and help the city optimize the traffic signals along important corridors. They detect unique identifiers of mobile devices, which are hashed (anonymized) and salted (anonymized differently each day), providing a unique identifier that can be paired among locations for the day. The sample data set also includes a corresponding timestamp, and sensor identification number in a format such as shown in Figure 1.

With these data, we can aggregate potential vehicle movements and discern signatures of cruising. However, this is not a straightforward process as there are some challenges in working with the data.

[1] Shoup, Donald C. "Cruising for parking." *Transport Policy* 13.6 (2006): 479-86. Web.
[2] Inrix. "Searching for Parking Costs Americans $73 Billion a Year | INRIX." *INRIX*. N.p., n.d. Web.

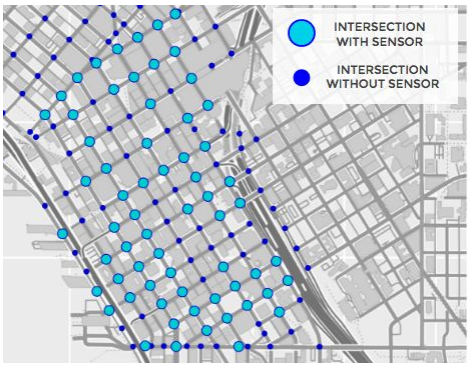
The sample data set includes approximately 200,000 observations per hour from 63 sensors distributed across the central business district, covering approximately 37% of the total intersections in the area as seen in Figure 2.
The data are highly noisy. There is a high prevalence of false positives (the sensor generates a reading that was not actually at the intersection, but rather a block or more away from the sensor) and false negatives (the sensor failed to read a device that actually did pass through the intersection).

Figure 1. Data example

DATA

HASHED MAC	TIME	SENSOR
KD98SDK8AHD8X	8:32:01	276105
8DJSKDLSX0WKK	8:32:01	276102
DQWPPOA09DSD	8:32:01	265402
KD98SDK8AHD8X	8:32:11	265302

Figure 2. Sensor network



DATABASE STORAGE

In order to work with such a large dataset and centralize our work, we decided to incorporate RethinkDB, a free and open-source database that stores JSON documents with dynamic schemas. We chose not use a relational database because the sensor reads, organized into trips by unique identifier and timestamp, would produce a very sparse dataset if arranged in tabular form. Therefore, RethinkDB was a perfect fit for our project as we needed to have a relatively free form with our data and be able to quickly aggregate by unique trip identifier.

SENSOR DETECTION RATE

An important step in our methodology was getting a better understanding of the data itself. One of our first interests was understanding sensor detection rate. To get an answer, we devised an experiment based on our existing data. We chose to focus on 2nd Avenue due to its high sensor coverage. First, we created a subset of trips that hit the starting sensor and hit the ending sensor without hitting any other sensor off of 2nd Avenue in between. Therefore, it was extremely likely that the vehicle travelled all the way down 2nd Avenue without deviating from the path. Given they hit the start and end sensor, the aggregate percentages of times were read by other sensors can be seen below in Figure 3.

This experiment showed us that there is high variability among sensor detection rates, with sensors such as 2nd and Pine performing well (it captured 58% of the vehicles that hit both the start and end sensor without deviating from 2nd Ave) and sensors such as 2nd and University performing poorly (it captured 22% of the devices that hit both the start and end sensor without deviating from 2nd Ave). As the rate of false negatives was relatively high overall, it was clear

that we would need to understand how to correctly route individuals on the most likely path traversed.

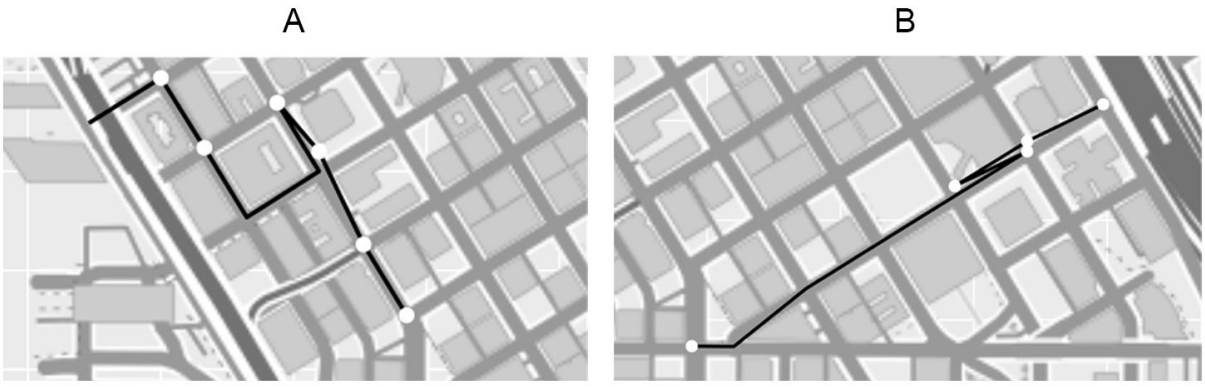
Figure 3. Sensor detection rate experiment

PATH CORRECTION

We observe several instances in the dataset where the same device is detected on neighboring sensors, either simultaneously or within a timespan short enough to make the journey between the sensors physically impossible. This causes two major problems:

1. 'Branching' of the path, i.e., a movement up and down a block not within the path of the unique traveler, as shown in Figure 4(A).
2. A back-and-forth movement within the path as shown in Figure 4(B).

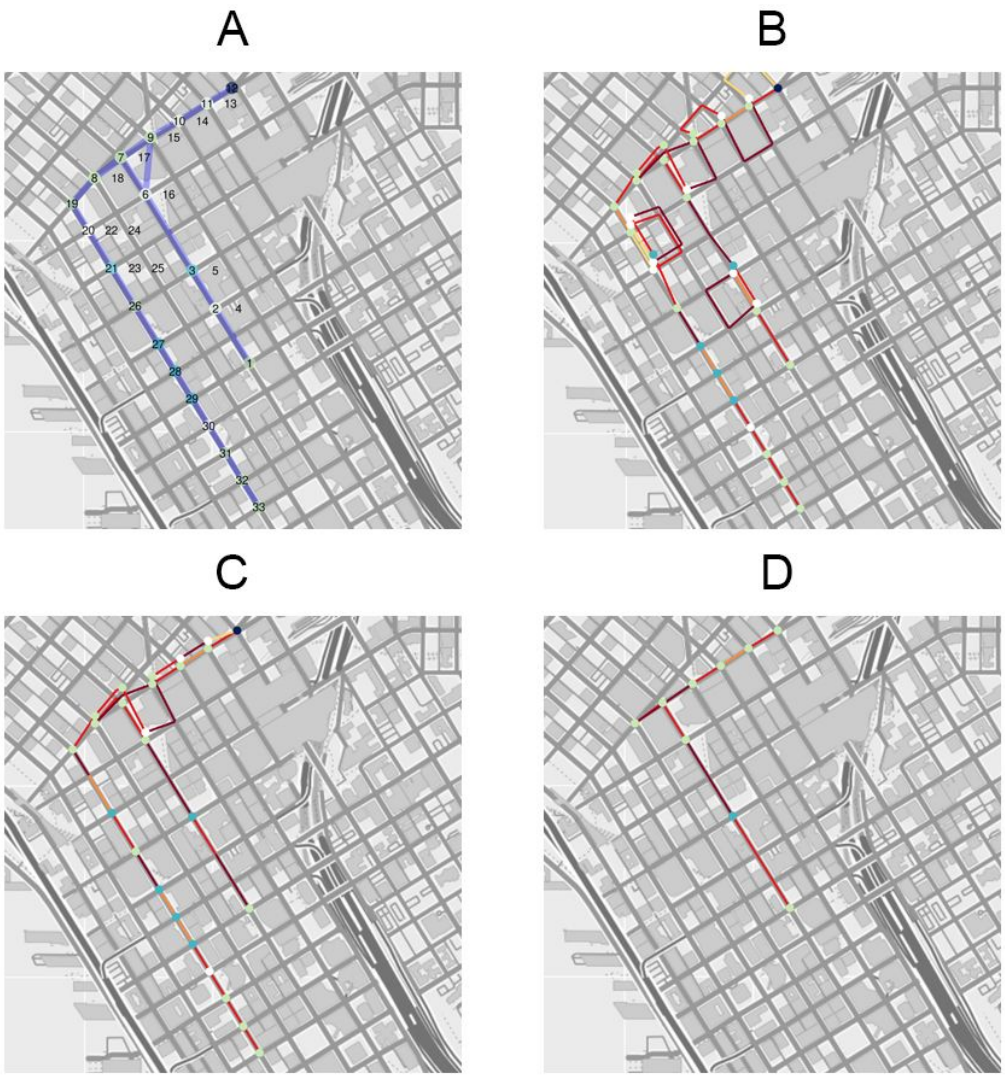
Figure 4. Routing artifacts due to high sensor range



We attribute this phenomenon to the range of the sensors, which extends for approximately one city block, and refer to the removal of these 'false positives' as prefiltering. We observe that the signal strength of the false positives is consistently lower than the detections made on the actual

path, which is the metric used for removing the false positives. We remove any detections of the same unique identifier address on different sensors within a timespan of three seconds. An example of a prefiltered path can be seen in Figure 5(B). Note that this path has been routed for the sake of visualizing the artifacts not accounted for by prefiltering.

Figure 5. Path correction sequence



After prefiltering, additional steps are taken to account for artifacts on time-scales larger than three seconds. Namely, if a vehicle is waiting at a traffic light between two sensors, it can still be detected by both, sometimes with an alternating pattern. If these alternating path segments occur on a two-way street, they produce the "back-and-forth" artifact of Figure 4(B). If they occur on a one-way street, the routing algorithm must direct the vehicle around the adjacent block to account for the apparent movement. This effect is called "looping" and can be seen in Figure 5(B). We repair this by summing the time spent at each alternating sensor, and editing the sensor IDs to match that of the sensor by which the traveler spent the most time. Similarly, pedestrians or cyclists moving against traffic on a one-way street will be routed onto side-streets, producing a zipper-like path (uppermost edge of the path in Figure 5(B)), which we repair by using an

undirected graph in place of a directed graph for trip segments deemed to involve walking or cycling versus driving (see the next section). Figure 5(C) demonstrates an example of a routed path after these corrections.

The sensors installed in downtown Seattle do not cover every intersection and varying sensor detection rates create unrealistic travel paths. To create realistic paths, Dijkstra's shortest path algorithm was used on a directed graph of downtown Seattle to fill in the most likely intersections crossed in a path. We extracted node (intersection) and edge information of downtown Seattle from OpenStreetMaps, inputted the information into NetworkX to create a directed graph, and created a lookup dictionary based on sensor to sensor shortest path computed from Dijkstra's algorithm. This lookup dictionary was used to fill additional intersection information in the paths, thus correcting the paths.

PATH SEGMENTATION

Three conditions require separation of travelers' paths into two or more segments. We refer to this procedure as segmentation. The first condition is a gap in sensor reads long enough to indicate that the traveler left the sensor grid. In this case, to consider their entire path as a single journey would be to artificially increase the likelihood of labeling it as an instance of cruising, given that return trips into the grid may cross prior paths and increase overall convolution. This type of error is particularly likely if travelers re-enter the grid at points other than those by which they exited. In such a case, segmentation is necessary to prevent the routing algorithm from connecting exit and reentry nodes by the shortest path through the grid. We have thus far designated gaps in reads of ten or more minutes as necessary segmentation points, though this designation requires further visual and programmatic testing.

The second type of segmentation pertains to trips containing stops of ten or more minutes. We assume the likelihood of a vehicle stopping in traffic for this duration is negligibly small. Therefore the likelihood is high that the traveler is either not inside a motor vehicle (and thus cannot be cruising), or they have parked. In either case, we split the overall trip into pre- and post-stop segments (see Figure 5(D)).

Depending on the eventual scheme for classifying trips as cruising or not cruising, we may choose to incorporate gap segmentation, stop segmentation, or both. A machine learning approach may very well benefit from the retention of stops within trips, as this could be a feature indicative of cruising (i.e. a wandering path followed by a stop could be an instance of cruising for parking, while a wandering path followed by straight-line movement at high speed is probably not). If instead we employ a mechanistic classification approach, it may be more straightforward to consider each instance of continuous driving separately. In either case, we have implemented an option for retaining the sensor reads from a stop on each of the segments after splitting.

The third type of segmentation involves separating trips at points where the mode of transportation likely changed. Sensors do not exclusively detect drivers, but pick up signals from walkers, bikers, and bus riders as well. We used velocity-based segmentation following path correction with Dijkstra's algorithm, to determine which paths segments were most likely

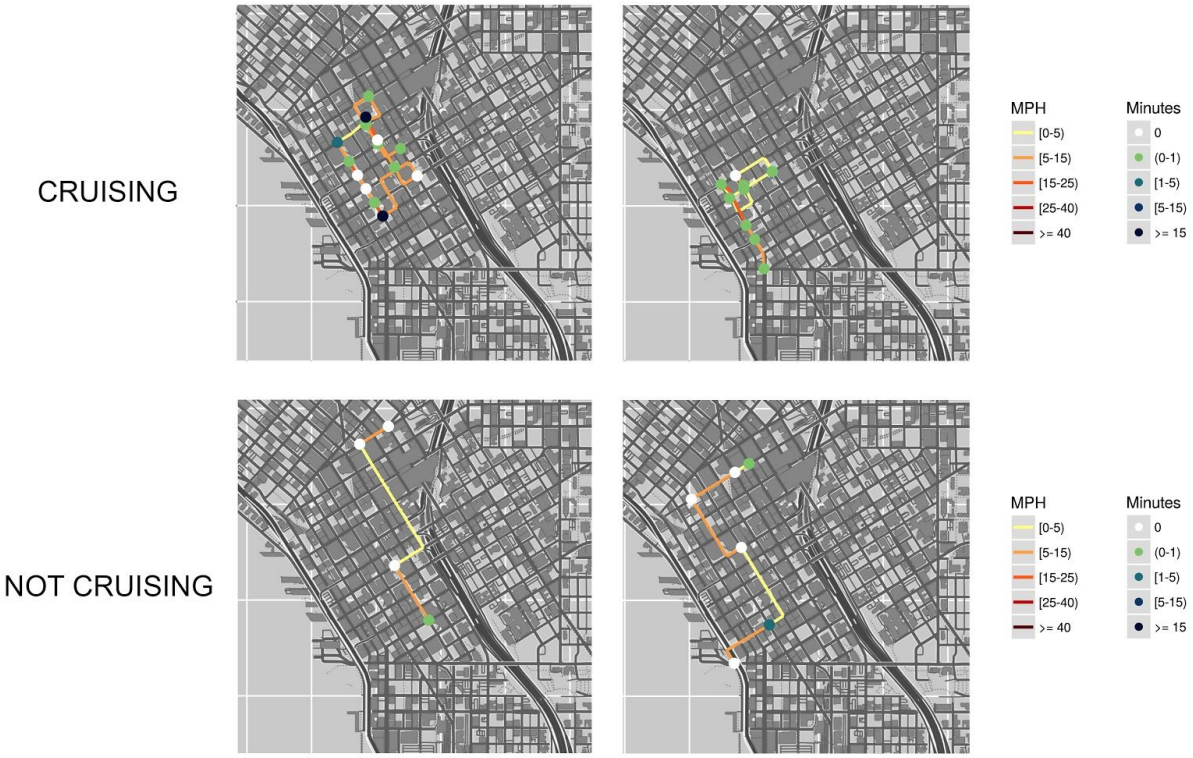
walkers versus drivers. When the majority of the path segments are believed to be walking, the paths are filtered out.

PRELIMINARY FINDINGS

CRUISING IDENTIFICATION

Through visualizing different paths of the unique travelers, we are able to see notable occurrences of cruising in Figure 6. Visual indications of cruising include, but are not limited to, paths looping over themselves, paths traveling only blocks away from the start position, and paths with high distance travelled but similar starting and ending points. These paths were indicative of individuals looking for parking and TNC circling throughout downtown Seattle. We label these paths and can use them for supervised learning training.

Figure 6. Visual identification of cruising and not cruising paths.



Additionally, metadata was generated for each of the unique paths. These data provide a useful vectorization of the paths and can be inputted into various machine learning algorithms. Table 1 contains a list of possible features to be used in cruising identification along with a description of the features.

Table 1. Features considered in cruising identification

Attribute Name	**Description**	**Purpose**
strength	Signal strength from sensor (decibels)	We will see if there is a difference in signal strength from drivers versus walkers which will help to identify the vehicles cruising.
velocity	Velocity traveled per path segment (meters per second)	It is possible that vehicles drive more slowly when looking for parking or looking to pick up a passenger.
sensor_hits	Total number of unique sensors hit in the trip	Vehicles cruising might hit a higher amount of unique sensors.
hit_total	The total number of times the intersection was hit in the path	We can see if the vehicle continues to come back to the same intersection.
trip_duration_time	Total trip time (seconds)	A longer trip duration more likely characteristic of cruising.
shortest_distance	Vincenty distance from start to end point (meters)	Short distance between start and end points more likely can show paths circling generally in the same area.
dijkstra_distance	Total distance of routed trip following Dijkstra's algorithm (meters)	We expect vehicles cruising to have a longer trip.
dijkstra_vs_vincenty_ratio	Vincenty distance from start to end point to Dijkstra's distance of overall path ratio	Cruising vehicles might have a higher Dijkstra-to-Vincenty ratio than average vehicles.
avg_speed	Average velocity of the trip (meters per second)	The average speed may be different for cruising cars vs. non-cruising cars.
max_speed	Maximum speed of the trip (meters per second)	The maximum speed will be different for pedestrians and vehicles.
avg_sensor_strength	Average sensor strength (decibels), where sensors exist	Sensor strength may differ for vehicles and pedestrians.
longest_stop_time	Longest stop duration (seconds)	Vehicles stopping may have actually parked.

SENSOR NETWORK

Overall, we have good reason to believe that the repurposing of the sensor network will lead to detection of cruising. To showcase this, we built a proof-of-concept heatmap visualization using the same data inputs needed for the final cruising heatmap. The proof-of-concept heatmap uses aggregated intersection-to-intersection traffic flow from our routed data and can be seen in Figure 7. To produce our final cruising heatmap, we need to only include paths that show signs of cruising, instead of all paths traversed, in the aggregation process.

Figure 7. Proof-of-concept heat map based on all routed data

NEXT STEPS

For this project, we have spent a large majority of our time transforming the data from mere sensor hits to most likely path traversed. Now that we have the latter, we can begin to think about implementing large-scale machine learning techniques for cruising identification.

One of the crucial steps in defining a cruising classification algorithm is to identify a useful representation for the space of paths. Many traditional classification and clustering algorithms are designed to work with vectors of fixed dimension and cannot be applied out of the box. In our setting we are working with time series, which can be either identified as a observations of a process on the sensor network, or a spatio-temporal process in the plane, and neither of those have standard distributions or metric structure to help us out. While there exist graph-based algorithms to detect loops in graphs (based on depth first search or the properties of the

adjacency matrix), cruising behavior which does not include passing through the same node is harder to detect through intrinsic graph properties and may require using the geospatial locations of the sensors. Further, graph clustering algorithms aim to cluster nodes of the graph, and in order to be adapted to paths, they require some notion of similarity of cruising paths. Similar is the situation with distance clustering approaches: we need to define a similarity measure which can work with paths of different lengths and can handle certain invariances, e.g. a clockwise loop should be similar to a path circling counterclockwise. A recent work by Bockholt et. al. explores ways to define similarity measures for paths in complex networks, and apply hierarchical clustering to classify paths taken by players of a board game. Unfortunately, for this application paths get clustered based on similar steps taken during the walk, which is not sufficient to identify cruising paths. Thus, further research is needed to discover cruising-distinguishing features, which can be achieved by evaluating a large set of features such as in Table 1, either through the careful study of the characteristics of a few cruising paths or by learning important features from a training set through variable selection algorithms.