

Real-time learning and prediction of public transit bus arrival times

DAVID NISSIMOFF – davidni@stanford.edu

Stanford University

12/16/2016

Abstract –This report outlines a method that was devised to leverage telemetry from public transit vehicles to learn patterns and provide real-time and accurate predictions of arrival times at arbitrary location along known routes. We also show results of applying the method to real-time data available from the bus transit system in São Paulo, Brazil. The results obtained are encouraging and demonstrate compelling prediction accuracy.

I. INTRODUCTION

Public bus services are still the prevailing form of transportation in many cities and millions of people depend on them every day to get to their destinations. In many places, however, the service is notorious for imprecise time schedules and large timing variations. While the system has an inherent dependency on external factors that it cannot control such as traffic and weather conditions, some improvements can be (and indeed have been) made at the infrastructural level in an attempt to mitigate the issues and provide a better experience for the riders. Several cities, for example, have started to deploy real-time monitoring systems that track the positions of each bus in its fleet and make that information available through the internet.

This report outlines the method that was devised for pre-processing the real-time vehicle telemetry into a form suitable for machine learning and then explains the techniques that were used to come up with accurate vehicle arrival time predictions from the pre-processed data.

The data used in the project comes from the bus services in Sao Paulo, Brazil, operated and managed by SPTrans (Sao Paulo Transport). SPTrans manages a fleet of more than 14,000 vehicles in over 1,300 routes [1] and many of the buses are equipped with the so-called AVL system – Automatic Vehicle Location [2] – which tracks the precise GPS coordinates of each bus roughly every 45 seconds and the data is made available through a public API branded “*Olho Vivo*” (Live Eye) [3].

II. INPUT DATA AND PRE-PROCESSING

At the core of this project lies a data collection pipeline which was built in C# and that is responsible for fetching real-time vehicle telemetry data, putting it through multiple pre-processing stages, and finally utilizing the results with machine learning techniques to come up

with predictions of when a vehicle will arrive at a given location. The following diagram illustrates some of the steps that will be described next.

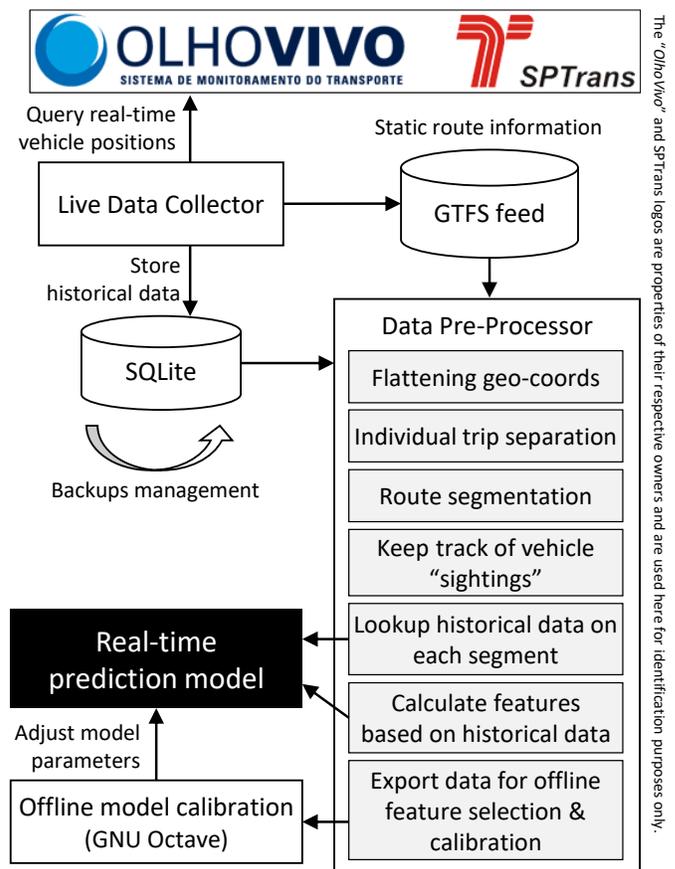


Figure 1 – Diagram of the processing pipeline

Raw input data

The input data comes from two distinct sources, both managed by SPTrans, which the pre-processing pipeline merges to facilitate the steps that come next. First, a GTFIS static database (General Transit Feed Specification: “a common format for public transportation schedules and associated geographic information” [4]). Second, the real-time “*Olho Vivo*” API that provides close to real-time information on the position of all active buses in a given transit line.

The “*Olho Vivo*” and SPTrans logos are properties of their respective owners and are used here for identification purposes only.

The real-time data consists of a time stream of tuples where each represents an observation of a vehicle in the fleet at a given time. Each tuple contains a timestamp of when the sample was taken, a unique vehicle identifier and the geographical coordinates of the vehicle at the time it was sampled. A peculiarity of the “*Olho Vivo*” API as of this writing is that it only provides access to the most recent sample for each active vehicle, and does not provide a precise timestamp of when the measurement was made. The missing timestamp information is a significant limitation. To overcome this, the adopted approach was to query the API with high frequency (roughly once every 2 seconds) and consider the response a new data point if it differs from the last seen value. The time the response was received from the API is used as an estimate of the measurement timestamp.

We also store the real-time data on a SQLite database for later usage. This was done so that the processing pipeline could be modified and improved incrementally, and the old data then replayed on top of the new pipeline. Data collection has been running day and night for over 50 days at the time of this writing and has amassed more than 13 million tuples from 1,693 distinct vehicles in 30 routes, occupying about 3.7GB of storage.

Flattening geo-coordinates

Working with geo-coordinates poses challenges that are best avoided. To that end, we leverage information from the GTFS database to transform the geo-coordinates onto a single new number that indicates how far along the route each vehicle has progressed along its route, in meters. In Cartesian coordinates, this is a simple geometrical problem of projecting a point onto a known path (polyline) and measuring its distance along the path from the origin. We therefore start by converting the geographical coordinates into approximate Cartesian coordinates as follows (Latitude and Longitude are assumed to be in radians below):

$$x = Longitude \cdot R \cos(Latitude)$$

$$y = Latitude \cdot R$$

$$R = 6,371,000 \text{ m}$$

This is a very good approximation for small distances, which is the case for the purposes of this project (each bus line stretches about 10 – 40 kilometers; very small compared to Earth’s radius). In Cartesian coordinates, we now search the entire route path, segment by segment, to find the one that is closest to the data-

point. The figure below illustrates how this is done for each segment. The distance to the closest segment is the distance from the point to the expected route. Data-points more than 50 meters away from the expected route are discarded. The result is a single number between zero and the route length that represents the position of the vehicle along the route.

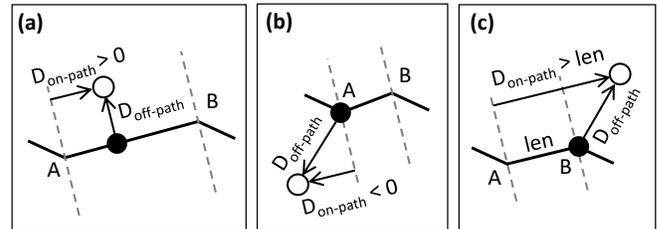


Figure 2 – Projecting a point onto segment AB. The projection of the hollow circle is shown by the filled black circle. In (a), the projection falls in the interior of segment AB; in (b), the projection falls in point A; in (c), the projection falls in point B

Individual trip separation

The next step was to split the dataset into individual trips (a trip is defined as an uninterrupted set of data-points from a vehicle as it goes on its way in a single direction). The following criteria were established for a set of data-points to be considered part of the same trip:

- All data-points must come from the same vehicle and transit line number;
- The time elapsed between adjacent data-points must not exceed 15 minutes;
- Position along the route must not go backwards (up to a threshold). Points that seem to go backwards by less than 600 meters are simply discarded and may be attributed to measurement jitter. If the total amount backwards exceeds 600 meters, the entire trip is discarded. This is done because in some cases the data offered by the SPTrans API was mislabeled and buses were on different routes than those reported, and this approach is an affective protection against such cases;

Line “875C-10-0” (*Lapa → Metrô S. Cruz*) was selected for evaluation of this strategy. This line covers about 19 km in length and has very high peak frequency (up to 15 buses per hour), thus offering large volumes of data (more than 640,000 points so far). Only 6.5% of the data had to be discarded as a result of applying the outlined criteria, and a total of 3,353 trips were counted during the first 28 days of data collection (an average of 120 per day).

Route segmentation

So far a trip is just a collection of data-points, each consisting of a timestamp and how far along the route the bus was at that time. This makes it non-trivial for establishing comparisons between different trips. The next step was, then, to discretize the route onto equally-sized segments, and register the timestamp and elapsed time of travel as a vehicle goes through each segment. This is done by linearly interpolating the bus position to estimate the time when the bus crossed from one segment to the next. The estimated time when the bus exited a segment is then subtracted from the estimated time when it entered the segment, to give the estimated time of travel for the segment.

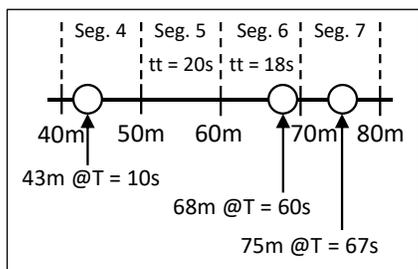


Figure 3 – Illustration of the trip segmentation procedure with hypothetical data. The 3 circles represent consecutive measurements of a vehicle as it goes on its trip. ‘tt’ is the calculated segment travel time

Through experimentation, 10 meter-segments were found to be a good compromise between resolution and computation overhead. In addition, we also ignore segments within 100 meters of the route extremums since it is not clear whether a bus in those areas is *en-route* or just maneuvering within the terminal in preparation for the next trip or entering the garage, stopping for maintenance, etc.

Data pre-processing summary

The steps described so far provide a framework to transform the raw observations of vehicle positions into a set of estimates of travel times through the equally-sized segments. We get one such estimate for each time a vehicle goes through a segment. Note that these are described as *estimates* and not measurements because we used linear interpolation to assess the position of the vehicles along the route in between measurements, and we are therefore estimating the exact time when the vehicle crossed from one segment to the next.

The remainder of the project uses the segment travel time estimates as the dataset from which predictions are made. Each segment is considered independently of all other segments, and therefore the notation is simpler if we think of the problem in terms of

S independent datasets, where S is the number of segments in the route. I.e.:

$$S = \left\lfloor \frac{RouteLength}{SegmentLength} \right\rfloor$$

For any segment $s, 0 \leq s < S$, we have multiple samples consisting of pairs $(ts_s^{(i)}, tt_s^{(i)})$ where:

- $ts_s^{(i)} \in \mathbb{R}$: ‘timestamp’ is the number of seconds since a reference constant date/time when the center of the segment was crossed for the i -th sample.
- $tt_s^{(i)} \in \mathbb{R}$: ‘travel time’ in seconds of the i -th sample
- Subscript s : the segment
- Superscript (i) : indicates the i -th sample for this segment

Note that the range of acceptable values for (i) varies by segment. For example, a vehicle that has completed half its trip at some time instant will have contributed segment travel time measurements for the first half but not the second half of the route. If we have collected n_s samples for segment s , then we have $1 \leq i \leq n_s$.

Also note that we make no distinction between different vehicles. Any vehicle that goes through a segment in a route contributes a new sample. This is reasonable for the problem at hand where the variance in travel times is attributed mainly to traffic conditions and not to individual vehicle performance / driver habits, and therefore all vehicles are considered equal.

III. THE PREDICTION MODEL

Now that we have information on individual segments in a workable form, we proceed to investigate how we can use them in order to predict the arrival time of buses at arbitrary places along the route.

The general idea is that we will seek a method of predicting the travel time of a segment at an arbitrary timestamp in the future. Since we know the position and timestamp where each active vehicle was last seen, we can then simulate its motion segment by segment, using the predicted travel time of each segment at the times when we expect the vehicle to be at that segment, yielding a good estimate of the vehicle’s future motion.

The problem at hand is, therefore, how to predict the travel time of a segment at an arbitrary timestamp. Figure 4 on the next page was generated from real data collected on a typical Monday morning and it helped

motivate the approach taken and which will be explained next. The figure shows how individual trips progressed over a period of 6 hours for a single bus line service. All times shown are local Brazilian time. This form of visualization was inspired on [12] and it is useful to investigate traffic build-ups. We can see one such occurrence at around the 8 km mark from around 7:00 am to 10:00 am and clearing up after that.

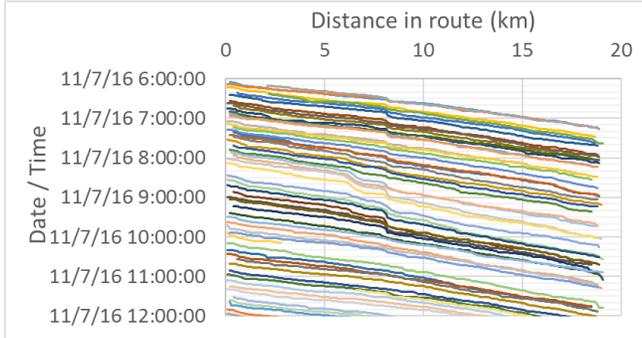


Figure 4 – A typical morning of the “875C-10-0” service. Each line shows a vehicle as it performs the ~19 km trip. Note the traffic jam at around the 8km mark in the early morning and that it clears up at around 10:00 am

We hypothesize that a good prediction model will include at least the following:

- Takes into account recent observations, since observations made at close timestamps tend to exhibit similar patterns. This is seen in the previous figure as adjacent trip appearing almost parallel to each other
- Daily periodicity: every day there is more traffic at 8:00 am than late at night at 11:30 pm.
- Weekly periodicity: Fridays perhaps exhibit more traffic than other days of the week.

One additional consideration is that business days also exhibit significantly different behavior than weekends and holidays. We therefore constrain our predictions and input samples to business days and ignore any samples collected on weekends or holidays observed in Sao Paulo.

We now proceed to analyze some of the methods that were explored for estimating the travel time of a segment at a ‘query time’ q using the sample pairs $(ts_s^{(i)}, tt_s^{(i)})$ as input.

Locally-weighted averaging

This is the simplest model that yielded acceptable results (weighted linear regression performed poorly as will be explained shortly). For ‘query time’ q , define

$$w_{s,i}^{(q)} = \exp\left(\frac{-(q - ts_s^{(i)})^2}{2\tau^2}\right)$$

Then the predicted travel time at timestamp ‘ q ’ is

$$\hat{tt}_s^{(q)} = \frac{\sum_{i=1}^{n_s} (w_{s,i}^{(q)} tt_s^{(i)})}{\sum_{i=1}^{n_s} w_{s,i}^{(q)}}$$

which is simply the weighted average of the input data with the given weights.

Modelling periodicity

We model periodicity of our data by transforming timestamps into a 3-D helix as follows. Given a timestamp t , we define the transformation:

$$v(t) = \left(\sin\left(\frac{2\pi t}{T}\right), \cos\left(\frac{2\pi t}{T}\right), \frac{pt}{T} \right) \in \mathbb{R}^3$$

where T is the model periodicity, and p is the helix pitch.

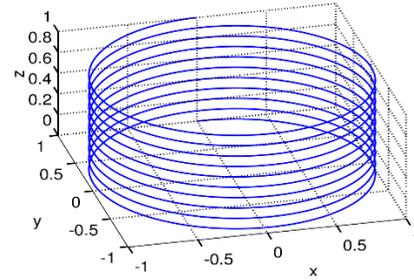


Figure 5 – 3-D helix visualization with $p = 0.1$, $T = 1$, t from 0 to 10

We then apply a similar method to the locally-weighted averaging just described, except the weights now use the transformed 3-D distance squared:

$$w_{s,i}^{(q)} = \exp\left(\frac{-\|v(q) - v(ts_s^{(i)})\|_2^2}{2\tau^2}\right)$$

The following chart shows how the weights are distributed for a particular set of parameter choices. The helix pitch ‘ p ’ defines the decay of the weights for older data.

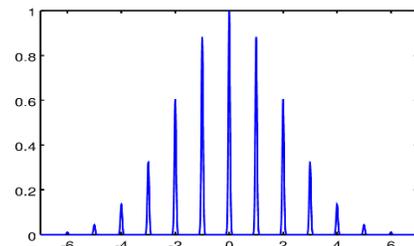


Figure 6 – Example of weights computed with helix transformation for query point $q = 0$, $T = 1$, $p = 0.1$, $\tau = 0.2$

Locally-weighted linear regression

The same that was just described for linear-weighted averaging applies equally to locally-weighted linear regression. This was not pursued further, however, because of the poor extrapolation performance of weighted-linear regression when there are few or no samples near the queried time q . Also, negative travel time estimates would be possible, which is not physically meaningful.

Choosing the locally-weighted averaging parameters

For any choice of T , τ , p , we can now obtain estimates of a segment travel time at any query time. Remember also that, as mentioned in the beginning of this section, our model should capture recent data, daily periodicity and weekly periodicity. Therefore it is not just a single set of parameters T , τ , p that we are after, but at least three such sets. This motivates approaching this as another machine learning problem. The input data for this is computed in the following way:

- Whenever a new vehicle position is obtained (a “sighting”), new segment travel times get calculated based on the actual vehicle motion (this proceeds as described on the “Route segmentation” section);
- We take all previous sightings of this vehicle along the same trip and, for each such sighting, compute what the estimated travel time of the newly-traversed segments travel times would have been using with 144 variations of parameters T , τ , p (9 values of τ , 8 values of p and 2 values of T). Importantly, the estimates are computed using only the segment travel time data that had been available at the time of that sighting.
- The set of estimates become a new feature vector, and we want to find a function that uses those features to predict the actual travel time of the newly-traversed segments.

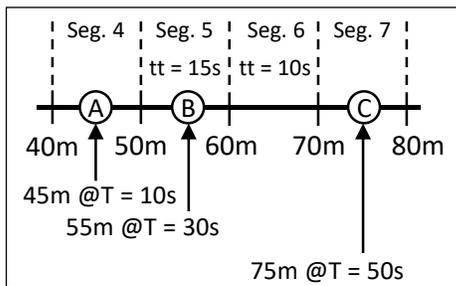


Figure 7 – Example of how data for the second learning problem is obtained, illustrating the progress of a vehicle to the right along a trip and 3 sightings A, B, C. When the vehicle is seen at C, two new segment travel times (‘tt’) are calculated (seg. 5 and seg. 6). Then estimations of those same times are made using only the data that was available when the vehicle was in A (and then in B) and with 144 variations of T , τ , p in each case.

Now we have a feature vector in \mathbb{R}^{144} . Because of the high computational complexity of this method (for each sighting, all 144 estimates are calculated from all previous sightings along the trip, resulting in $O(n^2)$ data points for n “sightings”), only a small percentage (0.1%) of the data points was used, still resulting in more than 30,000 training samples. Regularized linear regression using all features and $\lambda = 10^{-5}$ was found to perform well enough in this case and, using a train:test ratio of 70:30, attained $\sqrt{MSE_{train}} = 2.31$, $\sqrt{MSE_{test}} = 2.34$ seconds.

From the 144 features, we then selected the 3 best according to minimum test-MSE by brute force (this was done to avoid over-fitting and so that the real-time predictions only need to calculate the 3 estimates as opposed to all 144), obtaining $\sqrt{MSE_{train}} = 2.33$, $\sqrt{MSE_{test}} = 2.33$ seconds. The chart below shows the final results of prediction accuracy of the project when everything is put together. 80% of predictions are accurate to within 6 minutes even up to 1 hour into the future.

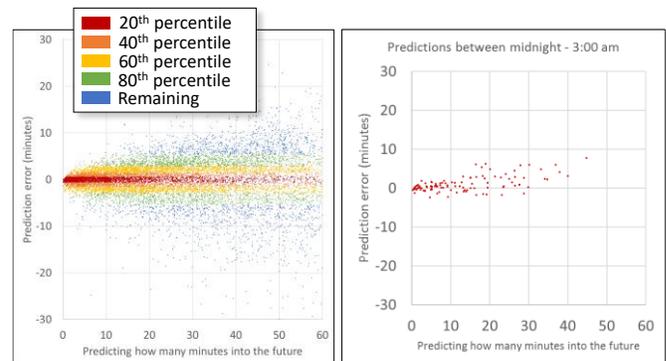


Figure 8 – Left: overall prediction accuracy of final project on bus line “875C-10-0”; right: data filtered to only show data points from late nights. There is much less variance in that case since there are virtually no traffic jams at those times. Best viewed in color.

IV. CONCLUSIONS

The main goals for the project have been achieved and it was demonstrated that it is indeed possible to predict bus arrival times based on past and real-time data using machine learning techniques.

Being able to predict where a bus will be up to one hour into the future is very powerful and opens up many interesting scenarios. Ideas for future work include migrating the implementation to a distributed cloud environment and building a user interface on top of this project to show useful and actionable data to a transit rider, as well as further improving the prediction models.

V. REFERENCES:

- [1] **SPTrans system indicators (“Indicadores do Sistema”)**. SPTrans. Available at <http://www.sptrans.com.br/indicadores/>
- [2] **Automatic Vehicle Location Technical and Functional Specifications (“Especificações técnicas e funcionais”)**. SPTrans. Available at http://www.sptrans.com.br/pdf/fretamento/especificacoes_tecnicas_e_funcionais_fretados_lei14971.pdf
- [3] **SPTrans Olho Vivo API**. Available at <http://www.sptrans.com.br/desenvolvedores/APIOlhoVivo.aspx>
- [4] **GTFS Static Overview**. Google Transit APIs. Available at <https://developers.google.com/transit/gtfs>
- [5] **Development of a real time bus arrival time prediction system under Indian traffic conditions**. Dr. VANAJAKSHI L. Available at https://coeut.iitm.ac.in/APTS_Finalreport_2016.pdf
- [6] **A Dynamic Prediction of Travel Time for Transit Vehicles in Brazil Using GPS Data**. GURMU Z. K. Available at http://essay.utwente.nl/59698/1/MA_thesis_Z_Gurmu.pdf
- [7] **Travel Time Prediction using Machine Learning**. MASIERO L. P., CASANOVA M. A., MARCELO T. M. de C. Available at <http://www.inf.puc-rio.br/~casanova/Publications/Papers/2011-Papers/2011-IWCTS.pdf>
- [8] **Online Grid-Based Dynamic Arrival Time Prediction Using GPS Locations**. NANDAN N. Available at <http://www.ijmlc.org/papers/372-C3023.pdf>
- [9] **Urban Bus Arrival Time Prediction: A Review of Computational Models**. ALTINKAYA M., ZONTUL M. Available at <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.683.2536&rep=rep1&type=pdf>
- [10] **Analysis and visualization of data on public bus transportation in Sao Paulo (“Análise e visualização de dados do transporte público de ônibus da cidade de São Paulo”)**. YAI, André K. Available at https://linux.ime.usp.br/~andreky/AndreYai_monografia_revisada.pdf
- [11] **Bus travel time prediction under high variability conditions**. REDDY K. K., KUMAR B. A., VANAJAKSHI L. Available at <http://www.currentscience.ac.in/Volumes/111/04/0700.pdf>
- [12] **A cellular automaton model for freeway traffic**. NAGEL K., Schreckenberg, M. Available at http://www.pd.infn.it/~agarfa/didattica/met_comp/lab_20140108/1992_origca.pdf
- [13] **Math.NET Numerics library for C#**. Available at <http://numerics.mathdotnet.com/>