Marius Latinis

Bus Arrival Time Prediction

Computer Science Tripos – Part II

Christ's College

March 25, 2017

Proforma

Name:	**Marius Latinis**
College:	**Christ's College**
Project Title:	**Bus Arrival Time Prediction**
Examination:	**Computer Science Tripos – Part II, July 2017**
Word Count:	**TODO(ml693): figure out**
Project Originator:	**Dr Richard Mortier**
Supervisor:	**Dr Richard Mortier**

Original Aims of the Project

Implement an algorithm which given the most recent GPS data predicts when the bus will arrive at the future stops. Evaluate the algorithm and show that on average it predicts with error less than 80s.

Work Completed

All that has been completed appears in this dissertation.

Special Difficulties

None

Declaration

I, Marius Latinis of Christ's College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed [signature]

Date March 25, 2017

Contents

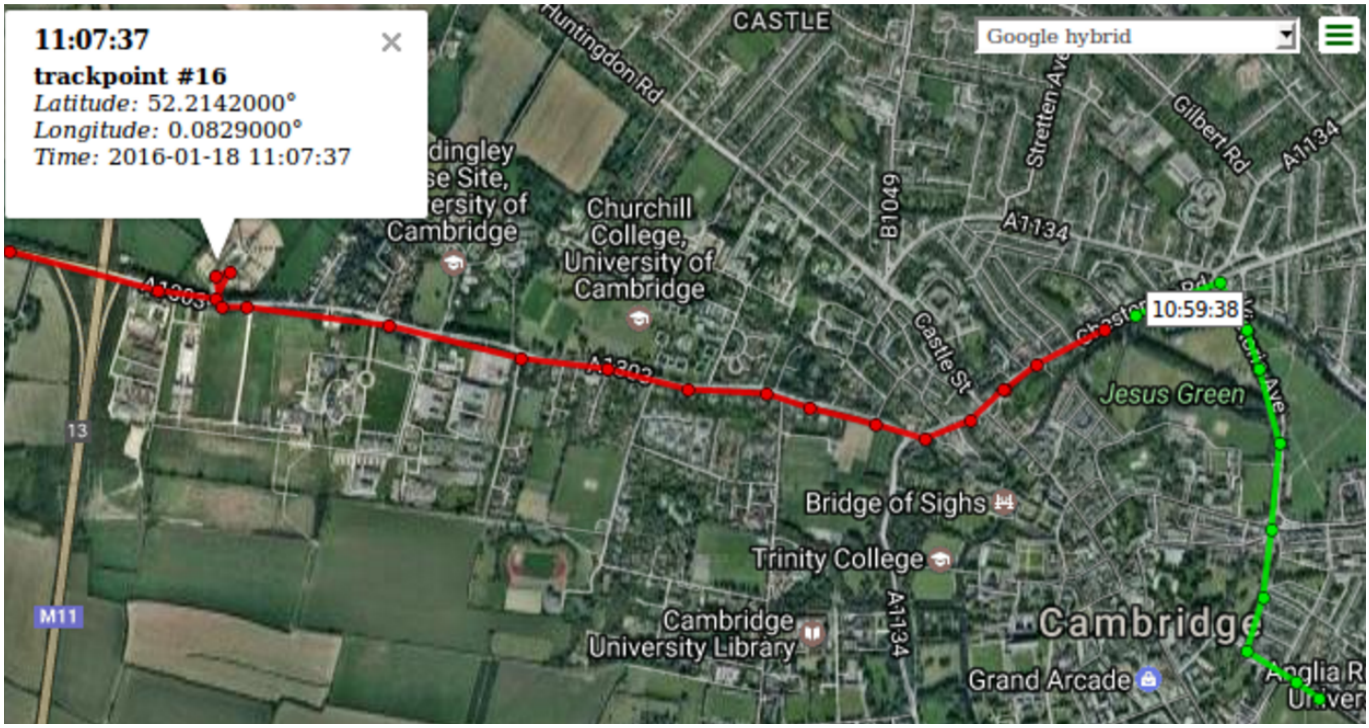
Chapter 1

Introduction

Motivation

In European countries public buses is a popular form of transportation. However, buses often arrive later than the timetable announces. Therefore, the project aims to build an algorithm which accuratelly predicts the bus arrival times based on the most recent GPS data.

Actual Problem Overview



The most recent GPS data (represented by green in the diagram above) shows where the bus has travelled up to now. The future data (represented by red) is not known during the prediction phase. My goal is to predict the future data given the present data.

In this specific example I want to predict when the bus will reach Madingley Park starting from Chesterton Road. One can see it takes about 8 min. (10:59–11:07) to travel such distance.

Chapter 2

Preparation

Starting Point

A company named Vix is sending real-time GPS data to Cambridge servers. The GPS data covers East-England bus companies TODO(ml693): which???. Cambridge converts the binary GPS data into a human readable JSON format. To start with, I was given 2 months JSON data (June and July 2016). That was the starting point data.

I started coding completely from scratch.

Preliminary Work

TODO(ml693): put something here

Chapter 3

Implementation

3.1 Data Preprocessing

Below is the starting point input data example:


```
"entities":[{"received_timestamp":1476477040,"vehicle_id":"122","label":"ATS-3603","latitude
":51.91048,"longitude":-0.5006834,"bearing":150.0,"timestamp":1470092134},{"received_timesta
mp":1476477040,"vehicle_id":"132","latitude":51.891663,"longitude":-0.45255125,"bearing":6.0
,"timestamp":1470091865},{"received_timestamp":1476477040,"vehicle_id":"187","latitude":51.4
93637,"longitude":-0.14694783,"bearing":282.0,"timestamp":1470092143},{"received_timestamp":
1476477040,"vehicle_id":"193","latitude":51.909676,"longitude":-0.5108846,"bearing":18.0,"ti
mestamp":1470092134},{"received_timestamp":1476477040,"vehicle_id":"194","latitude":51.88216
8,"longitude":-0.41519493,"bearing":114.0,"timestamp":1470092134},{"received_timestamp":1476
```


One file contains a single snapshot of all buses, with one entry per bus. An entry such as:


```
vehicle_id":"132","latitude":51.891663,"longitude":-0.45255125,"bearing":6.0,"timestamp":1470091865}
```


means that a vehicle nr. 56 was at geographical position $(51.88010, -0.42051104)$ on the 1st August 2016, GMT time 22:51:05[1].

A new snapshot file arrives once in 30s. The data preprocessing part takes all input files and converts them into another format:

File day15/bus300/trip2

```
time,latitude,longitude
2016-10-15 10:37:01,52.18485,0.20276
2016-10-15 10:37:32,52.18530,0.19788
2016-10-15 10:38:01,52.18548,0.19515
2016-10-15 10:38:32,52.18530,0.19343
2016-10-15 10:39:02,52.18557,0.19170
2016-10-15 10:39:32,52.18432,0.19070
2016-10-15 10:40:02,52.18271,0.19012
2016-10-15 10:40:32,52.18253,0.18840
2016-10-15 10:41:01,52.18297,0.18423
2016-10-15 10:41:31,52.18584,0.18509
2016-10-15 10:42:30,52.18647,0.18653
```

File day15/bus400/trip3

```
time,latitude,longitude
2016-10-15 12:01:35,52.20215,0.12460
2016-10-15 12:01:35,52.20215,0.12460
2016-10-15 12:02:35,52.19937,0.12705
2016-10-15 12:02:35,52.19937,0.12705
2016-10-15 12:03:35,52.19543,0.13064
2016-10-15 12:03:35,52.19543,0.13064
2016-10-15 12:04:34,52.19462,0.13423
2016-10-15 12:05:34,52.19444,0.13581
2016-10-15 12:06:35,52.19274,0.13581
2016-10-15 12:10:35,52.19247,0.13567
2016-10-15 12:11:35,52.19256,0.13409
```

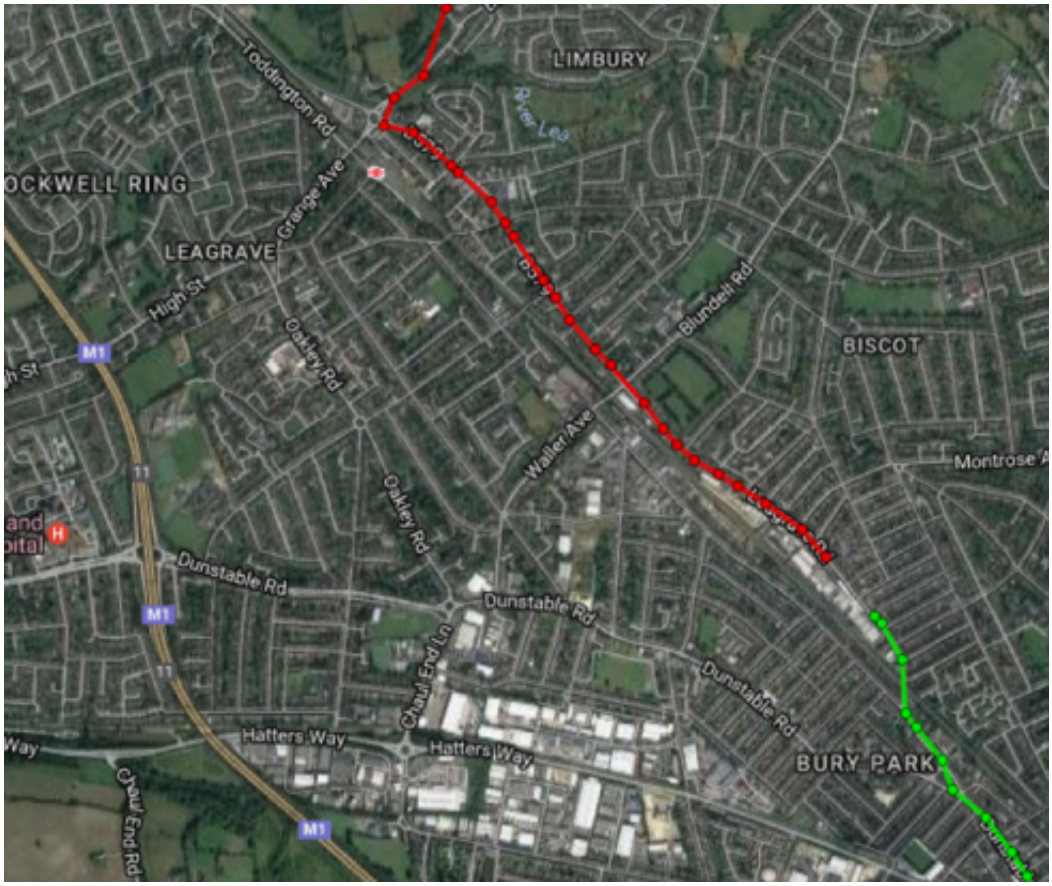
At the end I have a file for each trip made by one bus in one day[2].

[1] calculated from timestamp field

[2] note that a bus on a particurlar day could have made multiple trips

3.2 Route Detection Algorithm

3.2.1 Why an Algorithm is Needed



The GPS points indicated in green show how the bus has moved so far. The red points show where the bus will travel in the future, a path I do not know yet. Predicting when the bus will arrive at the stop requires knowing **where** a bus will travel next. How?

Approach Using Static Data

A standard way to find out where the bus will travel next is to know its route in advance. Some GPS entries contain additional fields, such as a **label**:


```
"vehicle_id":"122","label":"ATS-3603","latitude":51.90439,"longitude":-0.5070053,
```


An extra field **label** names the route a bus follows. Auxiliary data might indicate what route the label **ATS-3603** corresponds to. One can then look at the route and determine a sequence of stops a bus will visit next.

Unfortunately, this approach has drawbacks:

- Not all entries contain auxiliary fields. After all, the GPS device on the actual vehicle is responsible only for providing the latitude and longitude coordinates, not the route a bus follows.
- Good static data to infer routes based on additional fields is not straightforward to find. Field values might not match (e.g. a route named **SW-4** in the real-time data is actually **SWX-4**).

Buses are often changing routes[3] and data does not always indicate that.

- The static data only shows the sequence of bus stops, not a detailed sequence of streets a bus will follow. It is much better to know an actual path for accurate arrival time prediction.

New Approach Overview

To overcome the static data limitations, I propose a new way to infer the route a vehicle is following. I will align a sub-trip how the bus has moved so far with a historical path for which the route is known. The path aligning best to the current sub-trip will be used to predict where a bus travels next. To perform the alignment, I constructed an algorithm[4] that can tell whether one sequence of GPS points follows another.

3.2.2 *FollowsPath* Predicate Definition

I define a predicate $FollowsPath(sub\text{-}trip, path)$ to be true iff a *sub-trip* follows a *path*:

Two pictures show when I want the predicate to return true and when false. For the first example, the predicate returns true because it is intuitively clear that a red sub-trip follows a blue path. In the second example, the output is false, as the two GPS sequences do not intersect at all.

[3] a vehicle is unlikely to follow the same route every day

[4] from now on referred to as a boolean predicate

3.2.3 *FollowsPath* Predicate Implementation

Firstly, consider the case when a sub- trip's GPS points (P_0, P_1, ..., P_{n-1}) **exactly** follow another path:

Each point P_i is on some segment of another trip's path. For example, P_1 is on the segment AB. Let $x = |AP_1|$ and $y = |P_1B|$. Then the error[5] function defined as

$$err(P_1, AB) = \frac{x+y}{|AB|} - 1$$

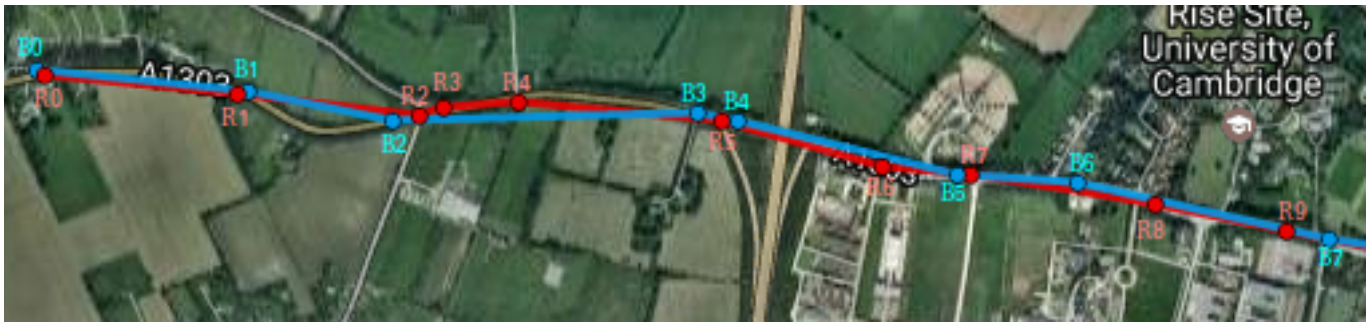
equals 0. For each point P_i, I find a segment S_{P_i} such that $err(P_i, S_{P_i})$ is minimised. The *FollowsPath* predicate returns true iff

$$S = \sum_{i=0}^{n-1} err(P_i, S_{P_i}) < n\epsilon$$

for a suitably chosen threshold value ϵ[6].

Note that $S = 0$ when a sub-trip follows a path exactly. Therefore, in this case, the *FollowsPath* predicate returns true, which is what I want.

The case when a sub-trip does follow a path, but not exactly (the real world situation), is illustrated in the following picture:



In this example I am aligning a sub-trip (R_0, ..., R_9) with a path (B_0, ..., B_7, ...). The computation of S proceeds as follows:

$$\begin{aligned} S = {} & err(R_0, B_0B_1) + err(R_1, B_0B_1) + err(R_2, B_2B_3) + err(R_3, B_2B_3) + err(R_4, B_2B_3) + \\ & err(R_5, B_3B_4) + err(R_6, B_4B_5) + err(R_7, B_5B_6) + err(R_8, B_6B_7) + err(R_9, B_6B_7) \\ = {} & (1.0294 - 1) + (1.0300 - 1) + (1.0021 - 1) + (1.0730 - 1) + (1.0208 - 1) + \\ & (1.0714 - 1) + (1.0286 - 1) + (1.0158 - 1) + (1.0126 - 1) + (1.0125 - 1) < 10 \cdot 0.3 = n\epsilon \end{aligned}$$

and the predicate again returns true, as desired.

Finally, when a sub-trip does not follow a path, triangle inequality will ensure large values of $err(P_i, S_{P_i})$[7], thus the sum S will exceed $n\epsilon$, and the predicate will return false.

The practical evidence that *FollowsPath* predicate works is shown in the evaluation section.

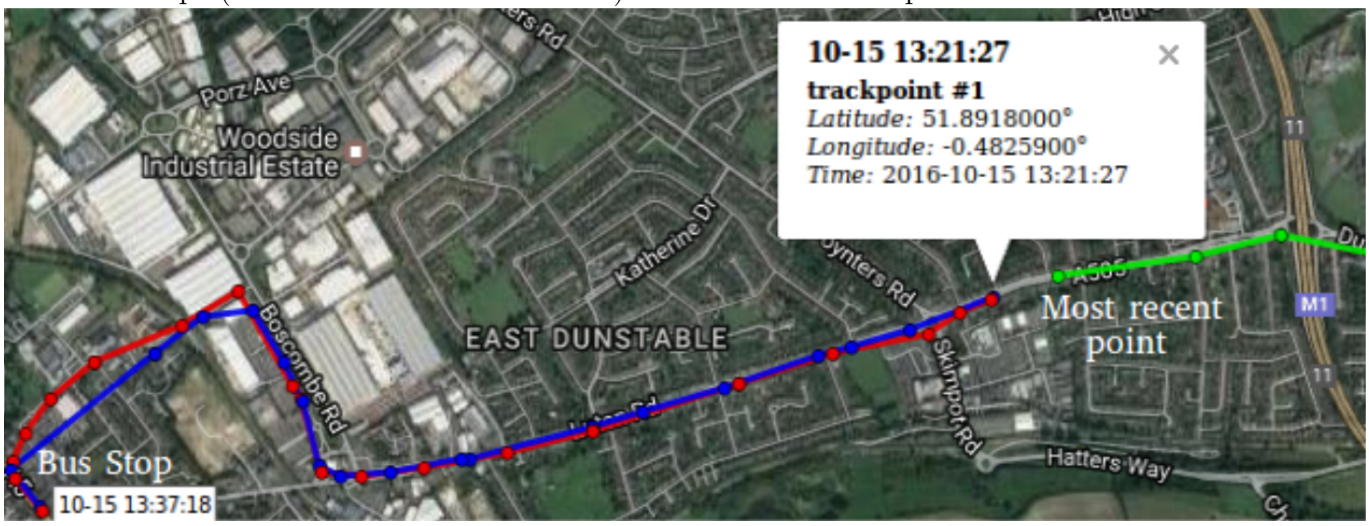
[5] error function takes a point and a segment as an input
[6] $\epsilon = 0.3$ is one choice
[7] $err(P_i, S_{P_i}) = \frac{x_i+y_i}{|S_{P_i}|} - 1 \gg (1+\epsilon) - 1 = \epsilon \implies S \gg n\epsilon$

3.3 Arrival Time Prediction

3.3.1 Basic Idea

The *FollowsPath* predicate allows to detect the route a bus is following. The route tells the next stops a bus will visit. This section shows how to predict the arrival time to each future bus stop.

Historical trips (which follow the same route) are used to make a prediction:



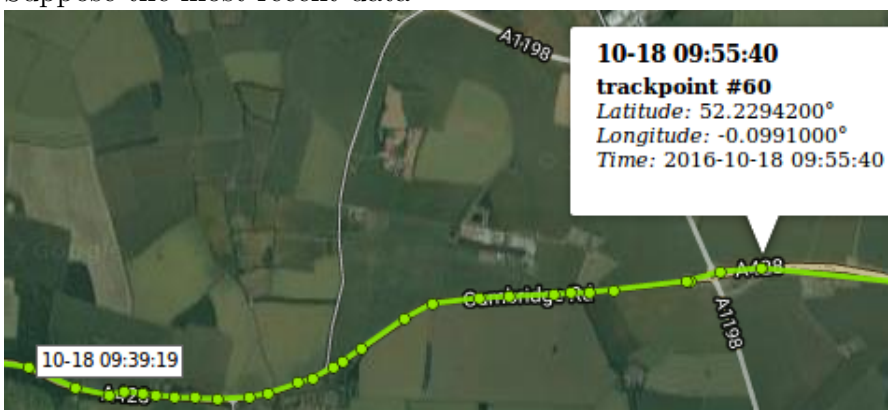
In the picture above 2 example historical sub-trips (blue and red) are shown. Both of them start where the vehicle currently located is (indicated by the most recent green point). Both of them represent the history how past vehicles travelled until the bus stop. I compute the duration how long it took for each historical trip to reach the bus stop. Knowing multiple duration values I return the median as the predicted arrival time.

3.3.2 Optimisation Nr. 1

If in the historical trips set certain trips have **just** travelled along the same route (e.g. in the last 30min.), I predict the median time just from those recent trips. Otherwise, the basic approach is used.

3.3.3 Optimisation Nr. 2

Suppose the most recent data



shows the distance a bus has travelled in the last 16 minutes. Certain historical trips

have travelled the same distance in a very different amount of time[8] (e.g. 4 min.). Predicting arrival time using these historical trips can be misleading. Hence I filter out such trips. I.E. I take only those historical trips into account that have travelled the same distance in roughly the same amount of time[9].

Optimisations are tested in the evaluation section.

3.4 Real-time System

TODO(ml693): finish implementing it

[8] this could have happened due different traffic congestion level in a different time of the day
[9] up to 30sec. difference is allowed

Chapter 4

Evaluation

4.1 Overview

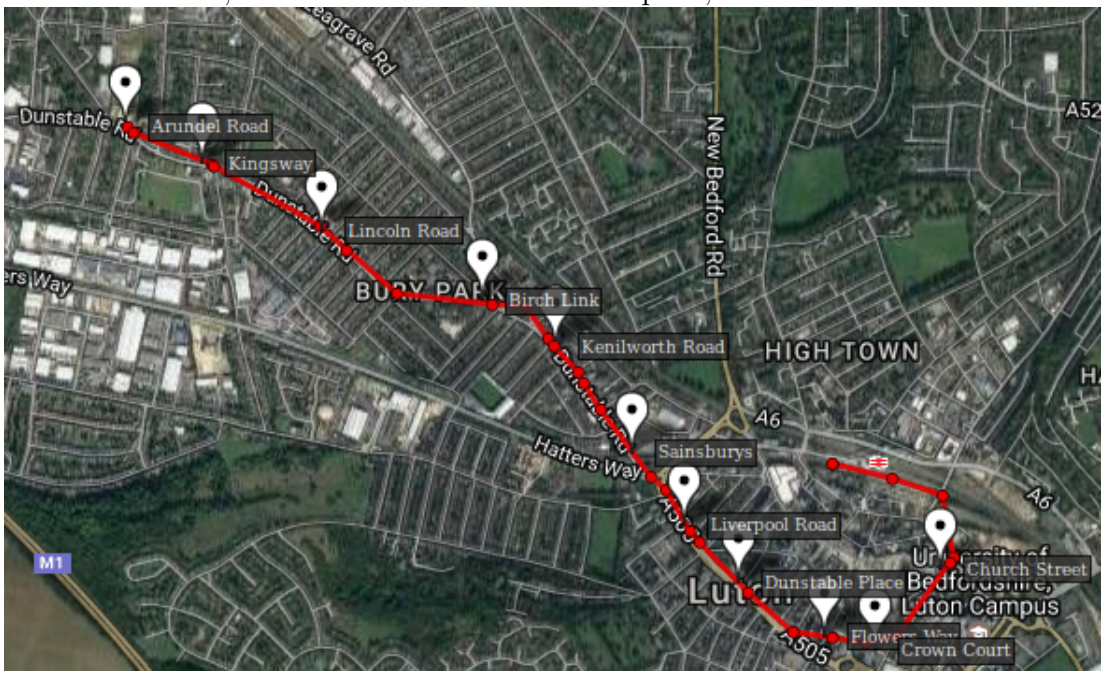
4.2 Route Detection Evaluation

I break the evaluation of route detection algorithm into multiple steps.

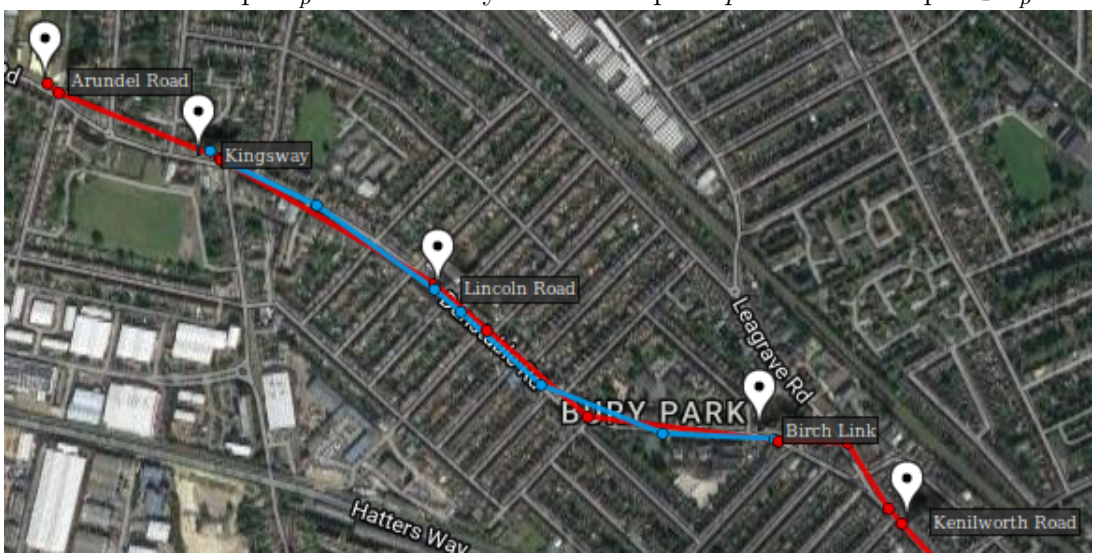
4.2.1 Sensitivity Score

The first step is to check whether the $FollowsPath(sub\text{-}trip, path)$ predicate returns *true* when the *sub-trip* indeed follows the *path*.

I have 10 routes, each route has an associated path, one of which is shown below[1]:



I have a set of trips T_p that actually follow the path p. For each trip $t \in T_p$ I take a *sub-trip*

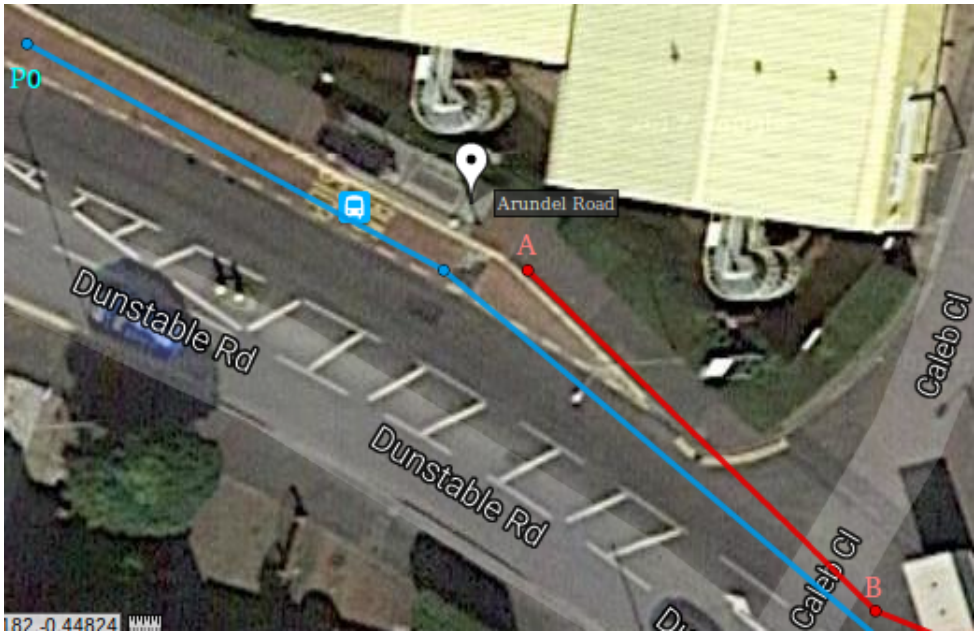


and check whether the $FollowsPath(sub\text{-}trip, path)$ predicate returns true or false.

If the *sub-trip* does **not** start from the first stop, the $FollowsPath$ predicate has 100% success rate for each route.

[1] Arundel Road on the top left is the first bus stop here

If the *sub-trip* starts from the first stop, the success rate drops significantly (and becomes unpredictable) due to the reason illustrated in the following picture:



One can see that $err(P_0, S_{P_0}) = \frac{|P_0A|+|P_0B|}{|AB|} - 1$ might become large. In fact, accumulated errors for the first few points (when the bus is not on any route yet) are so large the total sum exceeds $n\epsilon$ even if errors for the next points are small.

The conclusion is thus to discard the first few GPS points and start using $FollowsPath$ predicate after the bus has moved for a while and is following some route.

Chapter 5

Conclusion

I implemented an algorithm which gives sensible prediction results. The algorithm itself is simple, thus anyone coming after me to improve the system should not have trouble read and understand what has been done so far.

If started again, I would have worked in exactly the same way.

Appendix A

Links

1. Dynamic time warping is an algorithm for measuring similarity between two temporal sequences which may vary in speed. For instance, similarities in walking could be detected using DTW, even if one person was walking faster than another. I used DTW algorithm as a core part to implement my $FollowsPath$ predicate, with err function being used instead of a regular distance function $d(x, y)$. For more info look at

   https://en.wikipedia.org/wiki/Dynamic_time_warping

2. Arrival time prediction problem has been studied before. Two papers that I read and compared my project with are:

   - Bin Yu, William H.K. Lam, Mei Lam Tam: Bus arrival time prediction at bus stop with multiple routes

     http://www.sciencedirect.com/science/article/pii/S0968090X11000155

   - Pengfei Zhou, Yuanqing Zheng, Mo Li: How Long to Wait? Predicting Bus Arrival Time with Mobile Phone based Participatory Sensing

     http://www.ntu.edu.sg/home/limo/papers/sys012fp.pdf

3. Project code can be found at:

   https://github.com/ml693/bus

Appendix B

Detailed Evaluation Results

Appendix C

Project Proposal

(see next page)

Bus Arrival Time Prediction

Computer Science Tripos - Part II - Project Proposal

Marius Latinis (ml693@cam.ac.uk)

Project Originator: Dr. Richard Mortier
Project Supervisor: Dr. Richard Mortier
Director of Studies: Prof. Ian Leslie
Project Overseers: Dr. Markus Kuhn & Prof. Peter Sewell
Document Creation Date: 13 October 2016

Introduction

In European countries public buses is a popular form of transportation. However, buses often arrive later than announced at the stop. This annoys passengers and makes them complain. Wouldn't it be nice to have a good algorithm that predicts the bus arrival times precisely? This project targets such question.

Starting Point

Communication with buses streaming GPS data is already established. A file showing the GPS snapshot of 2 months (June and July) will be given to me as a starting data to work with. Real time bus GPS data will be presented for the optional extension. According to Dr. Lewis, the prepared and convenient bus GPS data files to work with are written in JSON format. They are roughly in one to one correspondence with the GTFS-realtime files, which are the files actually sent from the buses. Hence, I will work with JSON format as it is more readable and easier to process. That's the starting point data.

Work to be done

The project breaks down into the following parts:

1. Prepare maps for processing. A graph model of the map is considered in this project. The streets will be represented as edges with vertices being their intersections. One needs to **find** a map that contains streets covering the bus stops of consideration.

2. Prepare bus GPS data for processing. Bus GPS data also has to be prepared. The short description of what the bus data looks like according to Dr. Lewis is as follows:

> *"Buses announce their location data using GTFS-realtime format once every **30s**. GTFS-realtime format is then converted to a more readable JSON file. Each file contains information (current GPS location, unique bus identifier, bus route identifier and more) of around 1000 buses. "*

We hope that ***30s*** granularity will be enough to estimate the time it takes for a bus to travel from one intersection to the next intersection. Since it is unlikely that the bus will announce its data exactly at the intersection, we will have to interpolate the data.

The **core** project's part is to work only with buses in Cambridge and its neighbourhood. However, there might exist buses in the data that are out of the project's scope (i.e. a bus travelling to London). These will have to be filtered out.

3. Prediction algorithm implementation. Based on the data extracted in paragraphs (1) and (2), the algorithm has to inform when the next bus arrives to which stop. Suppose we are aiming to predict when a bus β being in a current location l_1 will reach the location l_n after having travelled through locations l_2, l_3, l_{n-1} in between (location is a vertex in a graph). For each i we ***average*** the most recent time $t_{i,\, i+1}$ taken for other buses to travel from l_i to l_{i+1}. Then the predicted time for β to reach l_n will be a sum $\sum_{i=1}^{n-1} t_{i,\, i+1}$.

Research concerning arrival time prediction has already been done in another paper[1]. The new prediction algorithm will differ in a few ways. Firstly, that paper essentially predicts a single value $t_{1,\, n}$ and outputs it as a result. Instead, we will calculate multiple intermediate values $t_{i,\, i+1}$, each of which will correspond to a predicted time for a bus to travel across one route's edge. We ***hypothesize*** that accurately predicting time for shorter segments and then summing all times up will lead to a better precision (read part 3 how the hypothesis will be tested). Secondly, the paper uses general machine learning models as a tool to predict arrival time. We will use calculations specific to the arrival prediction problem. That will ease the job of refining the algorithm in case it is not working well enough (see the first extension as an example how). It will also make the algorithm's optimisation task easier, as the algorithm itself targets a specific problem, rather than being a general ML tool.

// The pseudocode for the prediction algorithm

For each bus β streaming GPS data:

a) Extract l_1 - the current location of β

b) Look at β route to see the next locations $l_2, \ldots, l_n$

// The (c) part will be implemented following the summation idea above

c) Predict β arrival times $t_2, \ldots, t_n$ for each $l_2, \ldots, l_n$ and store these predictions in a multimap data structure *Bus[*β*] Arrives{(*l_2, t_2*), ... (*l_n, t_n*) }*

4. Prediction algorithm evaluation. Mathematical evaluation will be measured based on how well the algorithm predicts new arrival times. We will compare the prediction results with the actual time after we know when the bus has arrived. We want to use an evaluation metric similar to what's used elsewhere[2] (also known as the MAE_t value):

> *"In the trace-driven study, we [...] compare the predicted arrival time with the actual arrival time of the campus buses to compute the average of the absolute prediction error."*

Various error results are claimed (based on the bus distance to the stop) with ***80s*** being one of the values in that paper. Hence, we set a goal to build a model that would produce results with an error smaller than ***80s***. We will compute the MAE_t with $t_{1,n}$ being replaced by the sum $\sum_{i=1}^{n-1} t_{i,i+1}$. If the new MAE_t value is smaller than ***80s***, we claim that the **hypothesis** and hence the project was successful.

However, just assessing the project based on one number does not look convincing. To improve the assessment we propose to do a more in depth **quantitative evaluation**. Rather than computing a one global MAE_t value, we will also compute the MAE_t for each bus route and for each segment. In addition to that, we will count the total number of buses that arrived more than ***80s*** later, regardless of how late they arrived. All of these numbers will be reported in the progress report and final dissertation. The aim is to provide a few interesting metrics that will allow other researchers to have a choice in case they want to optimise for a particular one.

Finally, one more way to evaluate the project is by analysing how easily the algorithm can be run on a large scale. We will present this analysis in the dissertation by reporting its current complexity (w.r.t. the graph and JSON files size) and giving a clue what would be the sequential algorithm's part if run on multiple machines. That will provide others an idea of whether it is worthful to extend the project. However, **we would like to**

emphasize that scalability is not the main concern of the project, so going into too much detail here would be an overkill.

These four parts together form the core of the project.

Possible extensions

5. Optimise the prediction algorithm from the precision point of view. An advise how to do that was given by the overseer Dr. Markus Kuhn:

> "I would start with first estimating average edge times as a function of time of day and time of week, as load on the road network is usually a quite periodic process. I would then look into a textbook on standard multivariate statistical analysis for algorithm ideas on how to exploit statistical dependencies between spatially and temporally related edge traversal times. For example you could determine covariance matrices that describe how recently measured time intervals correlate with time intervals in the near future on either the same edge or on other nearby edges in your route graph. These averages and correlation matrices could then be used to calculate conditional probability distributions for traversal times in the near future, which could be added to make predictions. If you try to independently estimate the probability distribution of traversal time for each of thousands of edges, you may end up with quite noisy predictions, if you have more parameters to estimate than there are data points. This is where dimensionality-reduction techniques (such as principal component analysis) can become useful, where you try to represent the large vectors of edge traversal times that you try to predict as linear combinations of a much smaller number of base vectors (for principal component analysis, these will be eigenvectors of the covariance matrices), which still represent well the overall traffic situation in e.g. a town, but with far fewer parameters to estimate. Fewer parameters are easier to estimate from limited data, and hence allow faster updates."

6. Turning the static GPS data analysis into the real time processing. If our prediction algorithm gives promising results on the static data, extending it to work with the real time data would allow people to see when their desired bus actually arrives at the stop. This will involve writing a batch processing job that takes GPS files as input and continuously reports new arrival times as output. According to Dr. Lewis, the processing will be similar to the processing of stock transactions. Hence, we will split this task into the learning (i.e. surfing the net of how transactions are processed) and implementation parts.

7. Building a webpage to display prediction results. Given that we have predicted results, it would be nice to show them to the public. That's what the webpage will be used for. The aim is to have one stateless page where all the info is placed. Below is the example of the webpage content,

Station's Name	Next buses to arrive and arrival times
Covent Garden	No. 2 in 4min., No. 1 in 7min.
Leicester Square	No. 1 in 4min., No. 8 in 6min., No. 7 in 10min.

Resources required

The main resource required is "Bus GPS position data via Ian Lewis".

Other resources are my personal laptop running Ubuntu, GitHub, Google Docs, Hermes emailing system, SRCF.

Success Criteria

The project will be a success if I have implemented a working prediction algorithm that achieves ***error*** (*error* = average absolute error of real time prediction values*)* less than ***80s***. In addition to this main error value, we will plot an in depth **quantitative evaluation** statistics. Good results of other numbers will reinforce the main success criterion.

Timetable (planned starting date is 21/10/2011)

- Michaelmas term weeks 3 - 4: find maps open source data and make a graph model of it. A milestone for this part is to have a file containing a nice graph representation of Cambridge bus routes.

- Michaelmas term weeks 5 - 7: write a function which takes a JSON file as an input, extracts every bus with its GPS position, and for each bus finds the closest vertex where the bus currently is. One will also need to filter out those buses that

are present in the file but not in the project's scope (e.g. a bus travelling from Cambridge to London). Furthermore, some data might turn out to be garbage (e.g. bus presence along the wrong route) and this has to be taken into account. As it is important to prepare the main GPS data well, I am therefore giving 3 weeks for this part. This extra time will also help to rethink about the size of the graph once I started using it.

- Michaelmas term week 8: implement an "empty" framework. The framework will take JSON files and Cambridge map as an input and run the fake prediction algorithm on the input. The unimplemented fake part is (roughly) a function that takes a list of numbers as an input (the time it takes for other buses to travel across an edge) and produces a single number as an output (the predicted arrival time). The milestone here is to make sure that the framework itself is running, even if the results it displays are wrong.

- Christmas holidays weeks 1 - 3: replace the fake algorithm by the core prediction function. Evaluate the algorithm by computing the error numbers. If an error is too large, attempt to find a simple refinement of the core algorithm.

- Christmas holidays week 4: gather the best current algorithm statistics and write a progress report. **These results will be given for the midpoint presentation. Note that if everything is OK up to here, that means the core part of the project has been completed.** For the Lent term I am planning to work on extensions and am giving a further timetable:

- Lent weeks 1 - 3: work on optimising the algorithm from the precision point of view. One week can be spent reading about multivariate normal distribution, as it is likely to be used. Second week would be allocated for refining the algorithm. Third week would be spent comparing the new results with the best we have so far.
- Lent week 4: read how the stock data is processed in real time.
- Lent weeks 5 - 7: based on the info read implement the real time processing system.
- Lent week 8: create a webpage to display the results.

- Easter vacation: double check the work done before, write the first dissertation's draft.
- Easter term weeks 1 - 2: create the final dissertation's version.
- Easter term week 3: submit the dissertation.

Literature Appendix

1. Bus arrival time prediction at bus stop with multiple routes - Bin Yu , William H.K. Lam, Mei Lam Tam (http://www.sciencedirect.com/science/article/pii/S0968090X11000155)
2. How Long to Wait?: Predicting Bus Arrival Time with Mobile Phone based Participatory Sensing - Pengfei Zhou, Yuanqing Zheng, Mo Li (http://www.ntu.edu.sg/home/limo/papers/sys012fp.pdf)
3. Bus Based Probe Urban Travel Time Prediction - Wenjing Pu(https://books.google.co.uk/books?id=Tdps9w5jHKEC&pg=PA117&lpg=PA117&dq=multivariate+normal+distribution+for+bus+prediction&source=bl&ots=p5Aqeyo2NO&sig=Ml2XsklcIPBf9ChuR7ZdJLX2Qmc&hl=en&sa=X&ved=0ahUKEwj8soTzuenPAhVFBMAKHcC2CM8Q6AEIMzAD#v=onepage&q&f=false)
4. Applied Multivariate Statistical Analysis - R. Johnson, D. Wichern (https://www.pearsonhighered.com/program/Johnson-Applied-Multivariate-Statistical-Analysis-6th-Edition/PGM274834.html)
5. Multivariate Normal Distribution - Wikipedia (https://en.wikipedia.org/wiki/Multivariate_normal_distribution)