Optimal route prediction based on historical data collected using mobile nodes in urban areas

Konstantinos Chrysovalantis Manolios
University of Thessaly

Submitted to the
Department of Electrical and Computer Engineering
in partial fulfillment of the requirements for the degree of
Diploma of Science in Computer and Communication Engineering

October 2017

Main supervisor: **Spyros Lalis**

Co-supervisor: **Panagiota Tsompanopoulou**

Dedicated to

Aggeliki Manolios

In memory of

Mark Manolios

&

Lakis Paraskevas

Abstract

The goal of this work is to exploit opportunistically collected data about the time-stamped location of taxis in a city, based on this data to infer the speeds of taxis on different roads/road segments and for the different times of day/week, and then use this information to compute the best (fastest) route to follow between a source and destination in the city.

The main contribution of this work is the suitable processing of input data so as to produce good traffic data, and the exploitation of an existing routing engine to find piecewise shortest-paths. The available location data is transformed into a more compact form to enable faster analysis, and is split/processed separately for different time slots in order to capture the varying road traffic characteristics during the day and week. The shortest-path for a given time slot is found using available software. For trips which go beyond a single timeslot, we invoke the routing component for several consecutive timeslots, and then combine the results returned to derive a single recommended route.

Σύνοψη

Ο στόχος αυτής της εργασίας είναι η αξιοποίηση δεδομένων σχετικά με τις τοποθεσίες ταξί σε με πόλη, με βάση αυτά τα δεδομένα να παραχθούν συμπεράσματα σχετικά με τις ταχύτητες τον ταξί σε διαφορετικούς δρόμους η κομμάτια δρόμων και για διαφορετικές ώρες της ημέρας/εβδομάδας, και στη συνέχεια αυτή η πληροφορία να χρησιμοποιηθεί για τον υπολογισμό καλύτερων (συντομότερων) μονοπατιών που μπορεί να ακολουθήσει κάνεις μεταξύ μιας πηγής και ενός προορισμού σε μια πόλη.

Η κύρια συνεισφορά αυτής της δουλείας είναι η κατάλληλη επεξεργασία δεδομένων εισόδου με τέτοιο τρόπο ώστε να παραχθούν καλά δεδομένα κίνησης δρόμων, και η χρήση υπάρχοντών μηχανών δρομολόγησης για την εύρεση σύντομων μονοπατιών. Οι διαθέσιμες πληροφορίες τοποθεσίας αποθηκεύονται σε μια πιο συμπαγή μορφή, η οποία επιτρέπει γρηγορότερη ανάλυση, και είναι χωρισμένη σε διαφορετικά χρονικά παράθυρα για κάθε μέρα έτσι ώστε να μπορούν να εντοπιστούν τα ποικίλα χαρακτηρίστηκα κίνησης του δρόμου μέσα σε μια μέρα ή βδομάδα. Το συντομότερο μονοπάτι για ένα χρονικό παράθυρο ανακαλύπτεται κάνοντας χρήση διαθέσιμου λογισμικού. Για διαδρομές οι οποίες ξεπερνούν ένα συγκεκριμένο χρονικό παράθυρο, καλούμε το πρόγραμμα δρομολόγησης για τα συνεχόμενα χρονικά παράθυρα, συνδυάζοντας τα επιμέρους αποτελέσματα σε μια ενιαία προτεινόμενη διαδρομή.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Spyros Lalis for the past two wonderful years of having the chance to work with him. He has been a great source of energy, motivation and invaluable advice in times when all three were most needed. His professionalism and perfectionism have undoubtedly played a major role in my shaping into better engineers and people.

Secondly, I would like to thank Dr. Panagiota Tsompanopoulou for the great honor of being a member of my thesis committee, she too has played a major role through our studies, starting from our freshman year all the way to our final year. I would also like to take this opportunity to thank her for the chance of experiencing the “Numeric Analysis” lab from the scope of a lab assistant.

Special thanks to Beat (thebeat.co) for providing the data set, and more specifically to Dimosthenis Kaponis for his time and effort, which are highly appreciated.

Moreover, I would like to thank all my friends for all the support, group study sessions and fun times.

Additionally, I would like to thank my partner in most team projects, Alexandros Patras, for all the late-night coding and assignment sessions. He has been a great colleague in the good and bad times in this long 6-year journey.

Last but most importantly, I would like to thank my family for all their support and believing in me. More specifically, I would like to thank my mother, Aggeliki Manolios, for all her support during my studies, for the way she has brought me up and for everything she has provided for me up to now. Words are not enough to express my love and gratefulness towards her.

Contents

Chapter 1: Introduction 8

Chapter 2: System Overview 9

Chapter 3: Environment & Tools 11

3.1 Development Environment & Tools 11

3.2 Input Dataset 11

3.3 Terminology 12

3.4 OSRM 13

3.4.1 Match Service 13

3.4.2 Traffic Data for OSRM 14

3.4.3 Turn Penalty Data 14

Chapter 4: Implementation Details 16

4.1 Producing traffic data digests 16

4.2 Basic filtering before matching 17

4.3 Calculating the speed for each edge of a route 17

4.4 Calculating turn penalties 21

4.5 Computing routes over several timeslots 22

Chapter 5: Evaluation 24

5.1 Data Quality 24

5.2 Speed estimation and route recommendation 25

5.3 Route recommendation 27

Chapter 6: Related Work 29

Chapter 7: Conclusions and Future Work 30

References 31

Chapter 1: Introduction

Over the past decade, due to the rapid adoption of embedded sensors, smartphones and Web/Cloud based services combined with the ever-growing storage capacity of server machines, data can be more effortlessly acquired than ever before. Web/Cloud service providers are constantly exposed to extraordinary amounts of information, which not only give deep insight in areas where their very own services can be improved, but can prove to be invaluable for the production of new services and the speedup of research discoveries. A good example is the exploitation of search query data in order to detect influenza epidemics [1].

After the boom the web and web-based information, we are experiencing the rapid adoption of IoT (Internet of Things) devices. Embedded systems are beginning to infiltrate every single appliance, from light bulbs to fridges, vehicles, public transportation, wearables, health devices, security systems and many more. This creates a new world of interconnected sensing devices continuously producing more and more data at an ever-increasing rate. This data can prove even more valuable than data conventionally gathered through web sites and search engines. The real challenge is to take advantage and process it so as to produce truly valuable information.

As a concrete example, a large amount of data is nowadays produced by taxis equipped with GPS devices, which report their location as they roam between different areas in a city, round the clock. In fact, a relatively small number of taxis may be sufficient to cover the whole city. Among other things, this data can be used to solve the problem of efficient routing in the city. Routing is a daily challenge, people constantly attempt to navigate to the most efficient route as it can save a significant amount of time and decrease stress. Moreover, better routing can reduce fuel consumption, resulting in lower travel costs and less pollution.

This problem has been tackled using different approaches by individuals and organizations, including tech giants like Google and Microsoft. However, most of these approaches employ live sensor data streams. This thesis follows a different direction, and investigates a solution that relies only on frequently sampled historical data to provide quality routing suggestions. Quality suggestions could include less busy roads, routes containing fewer turns or routes with fewer stops.

The rest of the thesis is structured as follows. Chapter 2 provides a high-level overview of our approach. Chapter 3 summarizes the platforms and third-party tools used in our work. Chapter 4 describes the core structure of the system, providing details about how each functionality is implemented. Chapter 5 reports first evaluation results, based on the data set that was made available for this thesis. Finally, future work is discussed in Chapter 6.

Chapter 2: System Overview

Road networks in urban areas have been proven to display different variations of traffic flow, depending on the time of day, time of year and location. Moreover, traffic seems to be predictable during workdays, with an exception during summer months, when traffic becomes much lighter in bigger cities [2]. As a result, traffic cannot be treated as a whole, but instead each time of day has to be treated separately. To achieve this, we split our data in different timeslots. This way we can focus on times of day where routing is most needed, and since smaller timeslots require more processing, less busy times of day can be treated more loosely, using larger timeslots. Also, using a relatively small timeslot makes it possible to capture rapidly changing traffic situations, which is typical right before/after rush hours. In this work, we use 5-minute timeslots, but this is a configurable parameter and our approach can be easily tested for different timeslot values.

In our work, we estimate the traffic in a given road segment based on the respective vehicle speeds that have been sampled along that segment for the given timeslot. In turn, the speed of each vehicle is also inferred based on the locations reported at different points in time. The more samples available, the better the quality of the estimate. The sampling rate of the dataset provides high accuracy according to the exact route each taxi has followed.

Our system consists mainly of 3 key parts:

(i) The input service, which is fed with the raw input data and produces digested traffic data stored in database for easy further processing.

(ii) The profile generator service, which processes the data stored in the database and generates traffic profiles for the roads of the city for different timeslots.

(iii) The off-the-shelf routing service which returns the suggested (fast) routes for a given timeslot.

(iv) The part that puts piecewise results together into a single recommended route.

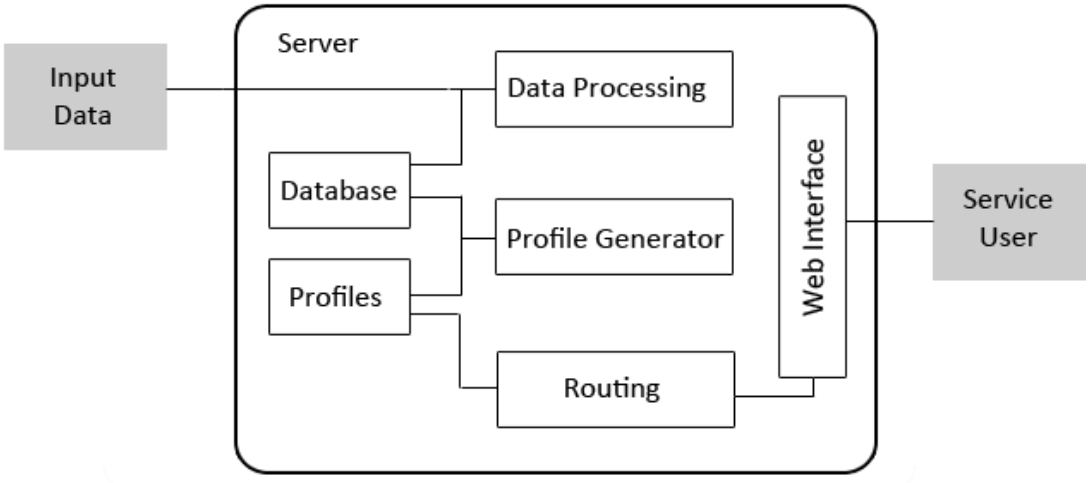

Figure 1. System Architecture.

The input service is invoked every time there is new traffic data available to produce updates. It receives the new dataset files as input, processes them in order to produce more lightweight digests/summaries of the traffic data, and stores the output in the system's database.

The profile service is invoked when new traffic data is available, or the processing method has changed. It retrieves from the database the lightweight digests/summaries that are generated by the input service, further processes them accordingly, and finally generates the profiles.

The route service produces routing suggestions for a trip requested by the user. In a nutshell, it finds the shortest path between the start and destination point in the city, based on the traffic data stored in the database. If the trip takes longer than a single timeslot, routing is performed separately for each consecutive timeslot, and the individual routing results are combined to produce the complete suggested route.

Chapter 3: Environment & Tools

3.1 Development Environment & Tools

- **VM:** Ubuntu 16.04 LTS
- **DB:** MongoDB (selected because of built-in support for geospatial indexing/queries)
- **Programming Languages:** Python
- **Open Street Map (OSM):** Collaborative project to create a free (open license) map of the world, based on volunteer work [3].
- **Open Source Routing Machine (OSRM):** Open source routing engine that runs using OSM map data. In our work, we use mainly its matching and routing services [4].
- **Leaflet & heatmap.js:** An open source JavaScript library for creating interactive maps, and a plugin for creating heatmaps [5][6].

3.2 Input Dataset

The dataset was made available to us by a private company. It contains location data from Lima, Peru, for a certain period, collected by more than 4500 distinct taxis with a very high sampling rate of a location measurements per 6 seconds.

The input data set consists of multiple CSV files. Each CSV file contains data in the following format, sorted in timestamp-order:

Field	Description
Timestamp	Date and Time of geo point retrieval. (ISO 8601)
Latitude	Latitude coordinate of data point.
Longitude	Longitude coordinate of data point.
Accuracy	Estimate accuracy of geo point.
Bearing	The direction of motion of a moving object described in degrees.
Driver ID	Anonymized ID of the driver of a particular vehicle.

Table 1 Input dataset format.

To visualize the data and get some feeling of data density for different areas in the city, we created a heatmap together with some scripts for generating simple quick statistics on the fly. Observing *Figure 3* the density of the data seems encouraging. This was also confirmed by our own checks on the dataset.

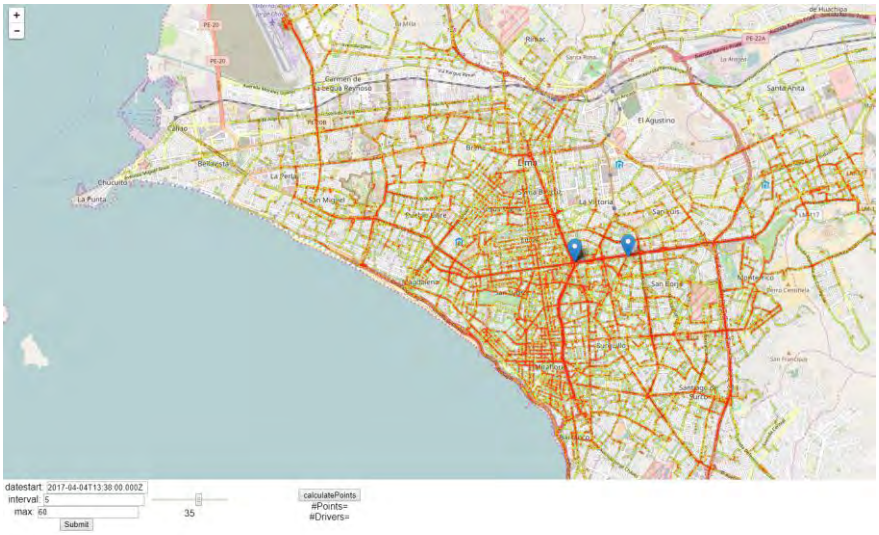

Figure 3. Heatmap created using Leaflet and heatmap.js.

3.3 Terminology

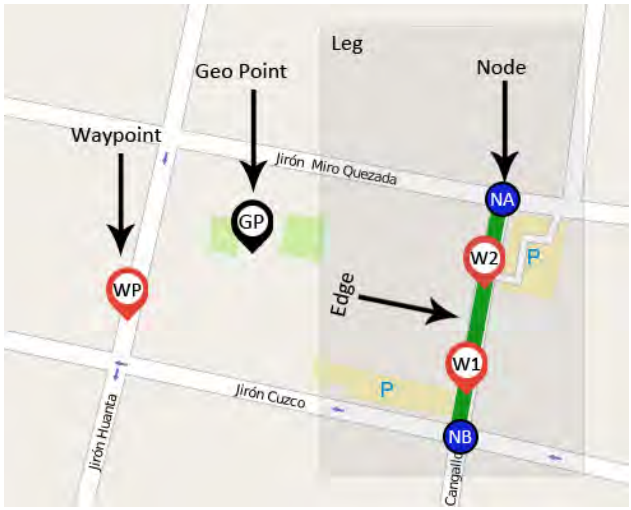

Figure 4. Terminology legend.

- **Geo point:** a pair of geographical latitude and longitude coordinates. Can lay anywhere on the map, not necessarily on a road.
- **Data point:** A data record of the input data set. Each data point includes a unique anonymized taxi/driver identifier, a timestamp and geo point.
- **Node:** A geo point in an OSM map with a unique identifier (NodeID). We are interested in nodes placed on roads, these nodes form a graph that represent the road network.
- **Edge:** A pair of nodes in an OSM map that are directly connected with each other along a road segment (there is no intermediate node between them along the same road).
- **Waypoint:** A geo point in an OSM map, to which an input data point is snapped (using the OSRM match service) so that it lies on a road.

- **Leg:** Connection between two waypoints. Contains everything that describes the movement between the two waypoints, like nodes/edges involved in the displacement from one waypoint to the other.
- **Route:** A series of waypoints. Used in navigation to describe the path from a starting waypoint to a destination waypoint.

3.4 OSRM

Routing on its own is a non-trivial task, it involves representing the road network as a directed graph, solving a shortest path problem with the use of one or multiple algorithms and a lot of complicated decision making. Fortunately, this problem has already been addressed and working solutions are readily available for calculating the best/fastest route between two locations in a city. Thus, instead of re-inventing the wheel, we rely on the built-in routing functionality of readily available free open-source software. Our software of choice is OSRM (Open Source Routing Machine). We run OSRM in a VM as a Docker image. For the map of Lima, we use a suitable subset of OpenStreetMap data.

3.4.1 Match Service

The OSRM match service helps implement some of our core functionalities in data input and processing. It can be used to snap multiple data points to an OSM map. More specifically, the service snaps each data point to a corresponding waypoint (that is a valid geo point which lies on a road), and returns the entire route.

The route that is retuned by the match service may comprise several legs. Each leg corresponds to the relation between two waypoints, and contains information regarding the directly-connected nodes (edges) that lie within these two waypoints, and the distance between the waypoints.

The match service returns a confidence level for the produced route. For our purposes, we request at least 85% confidence, else let OSRM reject the route. In any case, due to the high sampling rate of our input data (a location data point every 6 seconds, per each taxi driver) confidence is expected to be high.

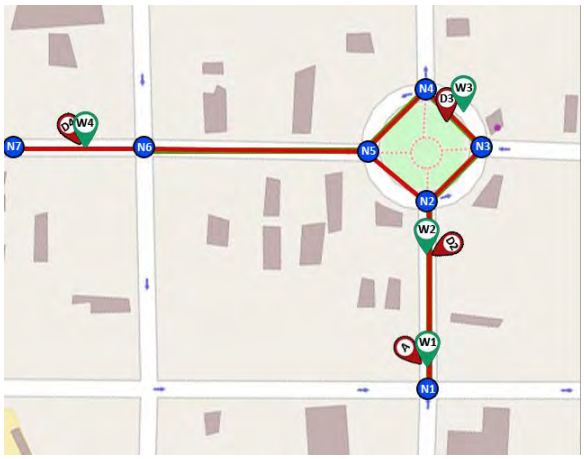

Figure 5. Information produced by the OSRM match service based on 4 input data points.

To give a concrete example, Figure 5 illustrates the information produced for four consecutive input data points of the same driver. The red pins denote the four input data points. The green pins denote the waypoints to which these input data points were snapped

by OSRM. The red lines denote edges. Finally, the blue circles denote the nearby nodes in the OSM map for the corresponding road segments.

To illustrate the importance of the confidence level for the OSRM match service, waypoints W3 and W4 have a significantly bigger distance than the rest of the sample, this could be caused by either a missing or a highly inaccurate data point. Following the snapping of data points to waypoints, OSRM has to determine the route followed to reach every waypoint. For waypoints W1, W2 and W3 there is no question regarding the route followed to arrive at each waypoint, thus confidence will be high, close to 100%. On the contrary, when analyzing the route from W3 to W4, one could argue that the driver could have taken a right turn at node N4, and followed a different path which concluded with a right turn at node N5 reaching waypoint W4. Since more assumptions (which may be false) have to be made regarding the route, the returned result will have a lower confidence level.

3.4.2 Traffic Data for OSRM

The internal data structures of OSRM used for routing can be initialized/updated by uploading CSV files with the following records, also referred to as traffic data:

Field	Description
Source (NodeID)	Node identifier corresponding to the starting point of the measurement.
Destination (NodeID)	Node identifier corresponding to the ending point of the measurement.
Speed (Km/h)	Traffic update speed value for the respective pair of nodes.

Table 2 Traffic update data record.

One of the problems that needed to be solved, is how to convert the location data samples into good speed estimates for the different road edges.

3.4.3 Turn Penalty Data

As part of the routing algorithm of OSRM, each time when the routing engine reaches a new node, it has to consider all the available continuation options for reaching the destination. These options may include multiple turns, which vary in sharpness. Merely considering the edge speed to each node after a turn is not enough, as the consequences of taking a turn would be completely ignored. However, in reality, the time it takes for a taxi to take a turn might be non-negligible, and when this cost aggregated over the entire route it can actually amount to a significant overhead, possibly making the route slower and/or unattractive compared to other options.

To give a simplified example, *Figure 6* illustrates a scenario where routing started from node N1, arrived at node N2, at which point it has to consider the next options, i.e., to move toward node N3 or node N4. Consider the case that the speed of the edge N2-N4 is significantly higher

than that of edge N2-N3, in order to conclude whether or not the turn is the better option, ones needs to be aware of the consequences of taking the turn, e.g., the time required to take the turn and the resulting speed reduction. Node N2 could be an intersection where vehicles are required to stop before proceeding, or a priority road where vehicles can proceed without reducing their speed. Clearly, each of these two cases translates to a very different turn penalty, which is critical for dependable routing results. Else, routing could return multiple faster zig-zag sub-routes, while in reality the total route would have a much longer travel duration than the sum of the time required for each sub-route.

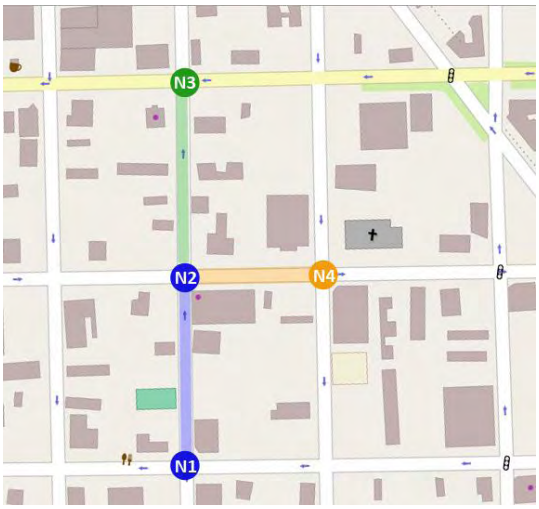

Figure 6. Turn penalty routing example.

To address this problem OSRM allows so-called turn penalties to be introduced in an explicit way. The internal data structures of OSRM used for turn penalties can be initialized/updated by uploading CSV files with the following records:

Field	Description
Source (NodeID)	Node identifier corresponding to the starting point of the measurement.
Via (NodeID)	Node identifier corresponding to the node located at the junction.
Destination (NodeID)	Node identifier corresponding to the end node indicating the direction of turn.
Penalty (s)	Turn penalty value in seconds, corresponding to turn described by the NodeID triplet.

Table 3 Turn Penalty Data Update Record.

Note that it is the user's responsibility to provide good values for the turn penalties. The OSRM routing engine simply uses the values provided, taking them into account when considering different routing options in the spirit of a typical shortest-path algorithm.

Chapter 4: Implementation Details

4.1 Producing traffic data digests

As an intermediate step, before we produce the actual traffic data to be fed into OSRM, we process input data in order to produce speed statistics for distinct timeslots, with the following format:

Field Name				Description
Date/Timeslot				Data/Timeslot stored as Day of Week (DoW), Week of Year (WoY), Timeslot of day (TS) and year (Y).
DoW	WoY	TS	Y	
Source NodeID				Node identifier corresponding to the starting point of the measurement.
Destination NodeID				Node identifier corresponding to the end point of the measurement.
Average Speed				Average speed of sampled vehicles.
Median Speed				Median speed of sampled vehicles.
Speed Variance				Speed Variance of sampled vehicles.
Min Speed				Minimum speed recorded while processing timeslot.
Max Speed				Maximum speed recorded while processing timeslot.
Nof Drivers				Number of Drivers that was included in this timeslot.

Table 4 Traffic data digest format.

Having this compact representation of speed statistics allows us to experiment with different heuristics for producing the actual traffic records for OSRM, without having to load and process the (large) input CSV data sets from scratch.

As explained previously, the reason we split data into different timeslots is to achieve better accuracy for transitional periods close to the rush hours of the day/week. Moreover, we store the date as DoW, WoY and Year for easier processing/grouping. For the purposes of our work, we decided to use 5-minute timeslots. This granularity is much coarser compared to the sampling rate, but also more than sufficient to capture the traffic dynamics in a modern city.

The process we adopt for producing the above records for each timeslot and edge (pair of direct Nodes) is briefly as follows:

(i) Initialize the speed dataset;

(ii) Load the respective input data from the CSV file(s) for the timeslot in question to corresponding driver ids;
(iii) Use the match service of OSRM to snap input data points to respective nodes and edges for each driver;
(iv) Calculate the speed and turn penalty for the found edges (again, make sure that this is clear to the reader) based on the timestamps of the input data points;
(v) Add the calculated speed and turn penalty to the speed data set for the respective edges;
(vi) When all input data has been processed, process the data sets in order to produce the speed record for each edge.

In the following sections we discuss each stage in more detail.

4.2 Basic filtering before matching

Even though OSRM takes care of outliers by itself, there are some cases where one can get a bad result due to one or more bad input data points. One such case is when a taxi is stationary, which means that consequent data points will have the same location coordinates. This creates a problem for the match service of OSRM, and might return a bad result. We need to feed the match service with as much valid points as possible to assist it in the detection of outliers. Another case is when OSRM fails to take care of some outliers in case data points appear to be too far away from one another. Finally, a taxi might be stationary but data points might have a slight difference.

To address many of these "anomalies", the distance between two data points is calculated before including the data points to the match service. If the distance is greater than 500 meters (provided that sampling rate is 6 seconds), the data point is rejected and we move on to the next one. The estimated distance is computed as (coordinates in radians) according to [7]:

$$est = 2 * \operatorname{asin}\left(\sqrt{\sin\left(\frac{lat2 - lat1}{2}\right)^2 + \cos(lat1) * \cos(lat2) * \sin\left(\frac{lon2 - lon1}{2}\right)^2}\right)$$

4.3 Calculating the speed for each edge of a route

When the match service of OSRM returns the result, we use them in combination with their timestamps to compute speed values, as follows:

$$speed = \frac{OSRM\ Distance\ (m)}{\text{timestamp2} - \text{timestamp1 (s)}}$$

Then, the calculated speed values are assigned to the corresponding edges. This is not a trivial as it may seem, and in fact it can be done in several different ways. Below, we discuss the options which we have considered, and motivate our choice.

Approach A: Closest Node Matching
Determine the coordinates of each node through OSM, compute the distance between each waypoint and nearby nodes, associate every waypoint with the closest node, and, finally, assign the speed that is compuated for each pair of waypoints to the correspondig edge(s) between the associated nodes.

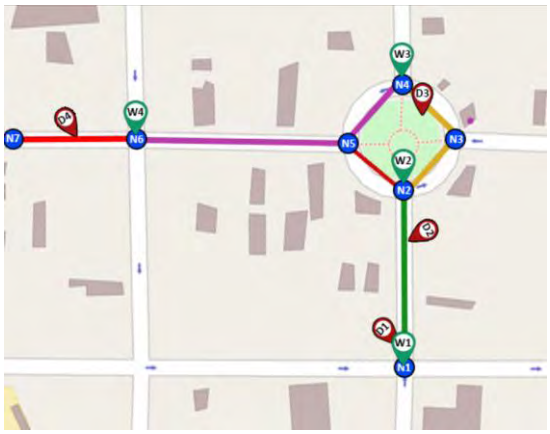

Figure 7. Approach A for way speed estimation for input datapoints D1,D2,D3,D4.

Figure 3 illustrates the approach, based on the snapping/route produced for the four input data points shown in Figure 7. Waypoints are attached to their respective closest node. Different edge line colors denode different speeds assigned to edges. No speed is assigned to red edges. we determine the speed for each pair of waypoints using the waypoints initial distance. In this case, we start by attaching waypoints to their closest nodes. Waypoint W1 is closer to node N1, waypoint W2 to node N2, waypoint W3 to node N4 and waypoint W4 to node N6. Next we analyze each pair of waypoints, starting from waypoint W1 to waypoint W2, since there is only one edge N1-N2 between the two nodes (best case scenario), we assign the computed speed to edge N1-N2. We then procceed to waypoints W2 and W3, where we find that there is no direct edge between N2-N4, istead we have edges N2-N3 and N3-N4, we will assign the same computed speed to both edges. Finally, in a similar manner we assign speed for edges N4-N5 and N5-N6 for waypoints W3 and W4.

Approach B: Total Edge Matching

Determine speed for every pair of waypoints and assign speed to every edge that includes these waypoints. Note that since some edges may be included in more than one pair of waypoints, some edges may end up with more than one speeds for them. As a solution, we can use the average as the final speed assigned to all edges. Alternatively, we could try to calculate the percentage of the speed that corresponds to each edge, but then we can use approach A instead.

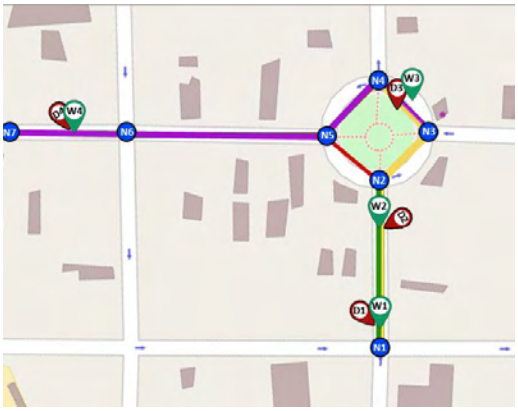

Figure 8. Approach B for edge speed estimation for input data points D1,D2,D3,D4.

Figure 8 illustrates the approach, based on the snapping/route produced for the four input data points shown in Figure 6. Different edge line colors denode different speeds assigned to edges. No speed is assigned to red edges. In this case, starting with the first pair of waypoints, W1 and W2, we find that only edge N1-N2 (green line) is assosciated with them, thus we assign to it the computed speed. Moving forward to waypoints W2 and W3, we find that edges N1-N2, N2-N3 and N3-N4 are the ones that include the two waypoints, we assign the computed speed to each one of these edges (yellow line), since edge N1-N2 is already assigned with a speed value from waypoints W1 and W2, we update the speed value as the average of the two speed values. Finally, proceeding with waypoints W3 and W4, we find that these lie within edges N3-N4, N4-N5, N5-N6 and N6-N7, and assign the computed speed to every edge (purple line). Again for edge N3-N4, we update the speed value assigning the average of the two speeds that have been computed for it.

Approach C: Minimum Full Edge Matching

Group waypoints in a way that at least one edge lies between the first and last waypoint, for each group of waypoints, calculate the speed between the first and last waypoint and assign the speed value to every edge that lies in between them.

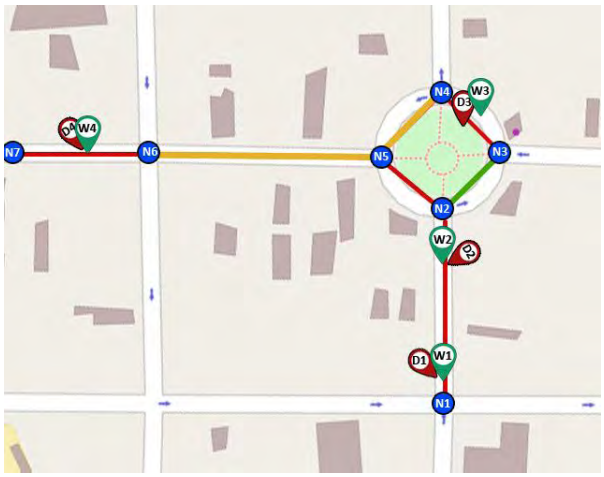

Figure 9. Approach C for edge speed estimation for input data points D1,D2,D3,D4.

Figure 9 illustrates the approach, based on the snapping/route produced for the four input data points shown in *Figure 6*. Different edge line colors denode different speeds assigned to edges. No speed is assigned to red edges. In this case we begin by grouping waypoints. Starting from waypoint W1, we attempt to group it with waypoint W2, since there is no full edge that lies n between these waypoints, we add waypoint W3 to the group. Checking again we note that full edge N2-N3 lies between these waypoints, we keep this as a valid group {W1,W2,W3}. Moving on, we start from waypoint W3 and we attempt to group it with waypoint W4. There is at least one full edge that lies in between them, thus, this is our second group {W3,W4}. Next we compute the speed for every group. For group {W1,W2,W3}, we compute the speed value from waypoint W1 to waypoint W3 and assign it to edge N2-N3 (green line). Finally for group {W3,W4} we compute the speed from waypoint W3 to waypoint W4 and assign the value to full edges N4-N5 and N5-N6.

Approach D: Rigorous Closest Edge Matching

Determine the coordinates of each node through OSM, compute the distance between each waypoint and nearby nodes, attach every waypoint to its closest node, requiring a maximum distance allowed between every waypoint and node, and, finally, assign the speed that is compuated for each pair of waypoints to the correspondig edge(s) between the associated nodes.

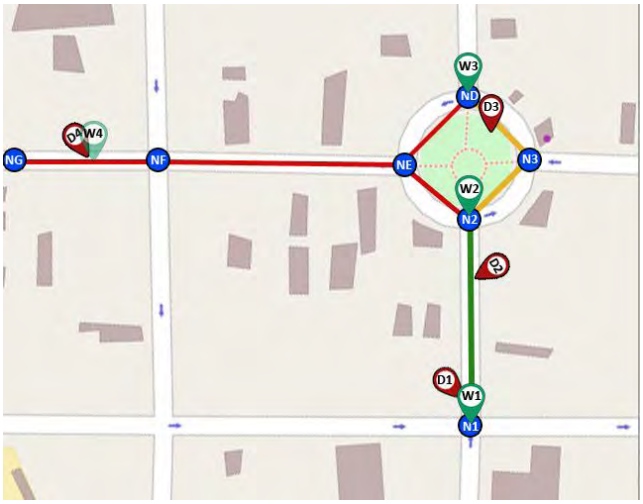

Figure 10. Approach D for way speed estimation for input data points D1,D2,D3,D4.

Figure 10 illustrates the approach, based on the snapping and route information produced for the four input data points shown in *Figure 6*. For illustration purposes, waypoints are snapped to their respectie closest node. Different edge line colors denode different speeds assigned to edges. No speed is assigned to red edges. In this case, we start by snapping waypoints to their closest nodes. Waypoint W1 is closer to node N1, waypoint W2 to node N2, waypoint W3 to node N34 and waypoint W4 is rejected as it is not close enough to any node. Next we analyze each pair of waypoints, starting from waypoint W1 to waypoint W2. Since there is only one edge N1-N2 between the two nodes (best case scenario), we assign the computed speed to that edge. We then procceed to waypoints W2 and W3, and find that there is no direct edge between them, instead we have edges N2-N3 and N3-N4 and assign the same computed speed to both edges.

Proposal

It seems that approach B is a good compromise. It requires the least amount of computing and should nevertheless lead to sufficient accuracy due to the large size and accuracy of the input data set.

4.4 Calculating turn penalties

Just like for traffic data, before we produce the turn penalty data to be fed into OSRM, we process input data so as to produce speed statistics for each timeslot, with the following format:

<table>
<tr><th colspan="4">Field</th><th>Description</th></tr>
<tr><td colspan="4">Date/Timeslot</td><td rowspan="2">Data/Timeslot stored as Day of Week (DoW), Week of Year (WoY), Timeslot of day (TS) and year (Y).</td></tr>
<tr><td>DoW</td><td>WoY</td><td>TS</td><td>Y</td></tr>
<tr><td colspan="4">Source NodeID</td><td>Node identifier corresponding to the starting point of the measurement.</td></tr>
<tr><td colspan="4">Via NodeID</td><td>Node identifier corresponding to the node located at the junction.</td></tr>
<tr><td colspan="4">Dest NodeID</td><td>Node identifier corresponding to the end node indicating the direction of turn.</td></tr>
<tr><td colspan="4">Penalty</td><td>Turn penalty value in seconds, corresponding to turn described by the NodeID triplet.</td></tr>
<tr><td colspan="4">Nof Drivers</td><td>Number of Drivers that was included in this timeslot.</td></tr>
</table>

Table 5 Turn penalties digest format.

The turn penalty values are calculated as follows. While the speed is calculated for each edge, we keep track of whether the vehicle was moving on the same road before this edge, or if it took a turn. For each different road where some vehicle recorded a turn to the specific edge, we keep a separate speed value. Once processing is finished, multiple speed values will be stored for the particular edge, with each value corresponding to a different possible turn that was recorded by a different vehicle. For each turn, we produce the turn penalty by comparing the turn speed to the value of the straight speed, as the difference in actual travel time between the two:

$$penalty = \left(\frac{distance\ (m)}{\text{turn speed (m/s)}} - \frac{distance\ (m)}{\text{straight speed (m/s)}} \right)$$

Obviously, the penalty for the straight (reference) route is set to 0. We should note that a turn penalty value could be negative, and thus be preferable to the straight route. For example, suppose a situation like the one in Figure 11, a vehicle has arrived at node A and has the options of either moving straight on the blue line, or turning right on the red line. In case both roads lead to the vehicles desired destination, taking the turn would actually be preferred, because moving straight involves stopping for a second time to cross the road before node B.

Figure 11. Turn at node A preferred over straight route at node B

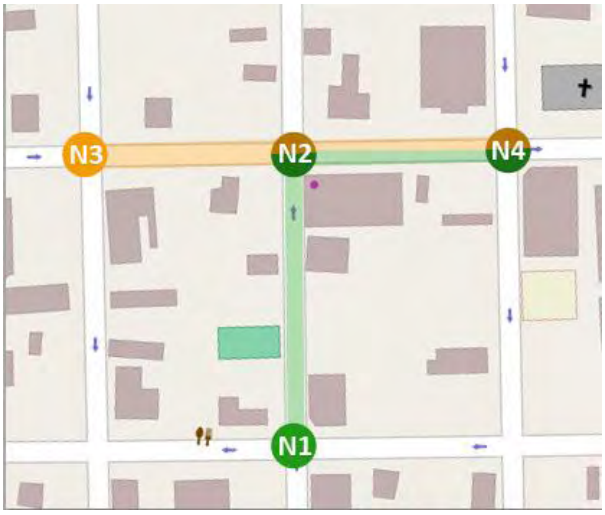

Figure 12. Producing turn penalty values for turn N1-N2-N4.

Figure 12 illustrates an example with two separate recordings from two drivers. Route N3-N2-N4 is the straight route. Let the speed that is calculated for edge N2-N4 via this route be 40 km/h. Route N1-N2-N4 is a route that includes a turn, leading to a second speed value for edge N2-N4, say 20 km/h. Granted that the distance between nodes N2 and N4 is 150m, the turn penalty value from node N1 to edge N2-N4 equals:

$$penalty = \left(\frac{0.15}{20} - \frac{0.15}{40}\right) * 3600 = \ 13{,}5s$$

In case the road for N1-N2 was bi-directional, there would be one more possible option, of turning left at node N2. Note that in the general case turn penalties are also direction specific, as the same penalty might not apply coming from the opposite direction on a bi-directional road.

4.5 Computing routes over several timeslots

While the route engine of OSRM solves the problem of finding the fastest path from a start location to a destination location, the suggested route is valid only for the specific timeslot that was used to produce the (profile) data fed into OSRM. In the general case, however, a trip will take a relatively long period of time (otherwise one would not need a ride in the first place), spanning over several consecutive timeslots (recall that we use relative fine-grained

timeslots of just 5 minutes). To find the desired route, OSRM has to be invoked several times for multiple consecutive timeslots using each time the proper profile data, until the destination is reached.

```
RR_Subroute(point_a, point_b, TS)
(1)   OSRM_Result = OSRM_Route(point_a , point_b, nof_alternatives)
(2)   (duration, subroute, new_point_a) = CheckDuration(OSRM_Result)
(3)   for i=0 To nof_alternatives
(4)     if duration[i] > TS then
(5)       newSubroute = RR_Subroute( new_point_a[i] , point_b, TS)
(6)       turnpenalty = RR_Turn_Penalty(new_point_a[i-1], new_point_a[i], newSubroute.point[0])
(7)       result[i] = subroute[i] + RR_Subroute( new_point_a[i] , point_b, TS)
(8)       Result[i].duration += turnpenalty
(9)     else
(10)      result[i] = subroute[i]
(11)  return result

RR_Route(point_a, point_b, TS)
(1)   result = OSRM_Subroute(point_a , point_b, nof_alternatives)
(2)   Sort(result)
(3)   return result
```

Pseudocode 1. Routing.

Pseudocode 1 gives a high-level description of the approach. More specifically, we employ a recursive approach whereby we start searching for routes in the next timeslot using as a starting point all the alternative routes returned for the previous timeslot. When combining together two sub-routes to produce a longer route, we also take into account the corresponding turn penalties, if any.

Note that in order to compute a globally optimal complex route, it is necessary to take into account all possible routes for a given timeslot, rather than just the fastest ones. This is because the optimal route may start with a very slow sub-route which then leads to very fast routes in the next timeslots. Of course, this increases the complexity of the computation. To reduce complexity, an alternative option is to keep each time only the *k* best routes, and use them as starting points for the next timeslot. However, especially for small values of *k*, this may lead to arbitrarily bad solutions.

Chapter 5: Evaluation

5.1 Data Quality

We performed a statistic analysis regarding a particular timeslot corresponding to an active traffic period, on a Monday from 14:10 to 14:15.

Due to the way taxis operate, we expected to have a high percentage of stationary samples. Indeed, Figure 13 shows that 33% percent of the samples were stationary.

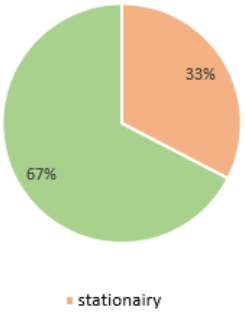

Figure 13. Stationary sample percentage.

Table 1 shows that out of the total 50974 edges of the city road network, the majority have samples for just 1 or 2 unique drivers, while only a few have data coming from more than 20 different drivers. To have a better feeling of the percentages, Figure 14 ??? shows the same information in the form of a pie chart. The gray areas parts of the chart correspond to edges with bad sampling, whereas greener areas correspond to edges with adequate samples. Speed variance was 13.26.

drivers	edges
1	12365
2	10225
3-5	10793
6-10	4122
11-20	1557
>20	303
total	50974

Table 6. Edges per unique driver number.

Figure 14. Edges per unique driver number percentage.

Finally, in *Figure 15* we can see the amount of OSRM match requests accepted with a confidence requirement of 85%. BadReq represents bad requests that could not

return any match at all, which account for just 3% of all total requests. About 22% of the matching requests was rejected due to low confidence. This is an area where the import/match algorithm could possibly be improved.

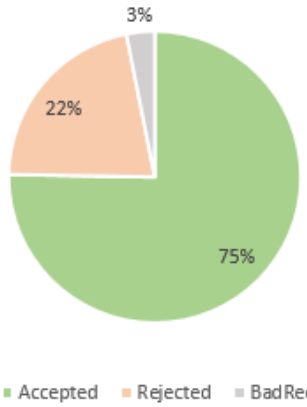

Figure 15. OSRM match response status.

5.2 Speed estimation and route recommendation

In order to perform a sanity check on the traffic data produced, we split the data set into 2 parts, the first is used to produce traffic data, and the second one is used to simulate current traffic conditions. To achieve this we wrote a script that browses through drivers and finds drivers who are on the move on roads for which we have traffic data. Once a driver on the move is found, we calculate our estimate time of arrival using our traffic data and then compare it to the actual arrival time.

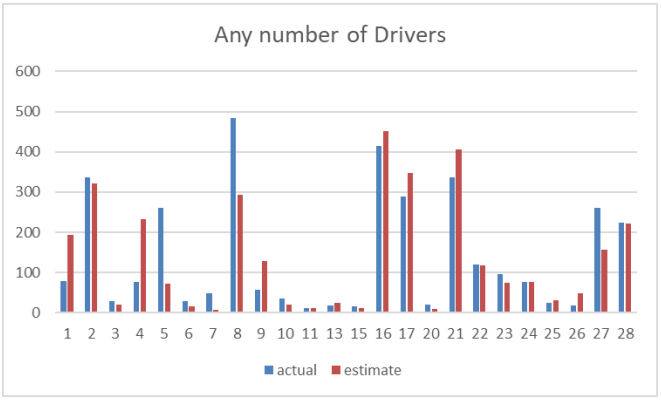

Figure 16. Actual vs estimated travel time using traffic data.

Figure 16 had many more samples than the other simulation, because more traffic data was available for more roads, though this was not quality data thus estimates sometimes have large errors. Sample normally had 400 routes, 25 were randomly picked to be illustrated.

Figure 17 plots a sample of at least 5 drivers result was much smaller and driving times that corresponded to data was smaller, but we can see that there are far fewer errors than in the case of the full traffic data case. Y-Axis scale is smaller so error is a bit smaller for most samples.

We have to keep in mind that test was performed with a simple script using data from the other half of our dataset. This means that some taxi driver could be moving slower for some reason. More drivers certainly make the sample look more stable

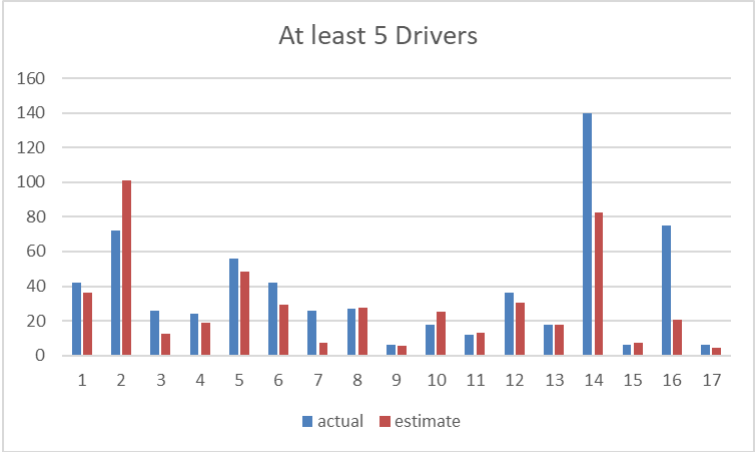

Figure 17. Actual vs Estimate Speeds using traffic data by at least 5 drivers.

Figure 18 illustrates some cases where the simulation exceeds the timeslots. In sample number 1 estimation ended before the need to switch like in sample 2 and 5, instead in samples 3 and 6 estimation was longer than actual duration.

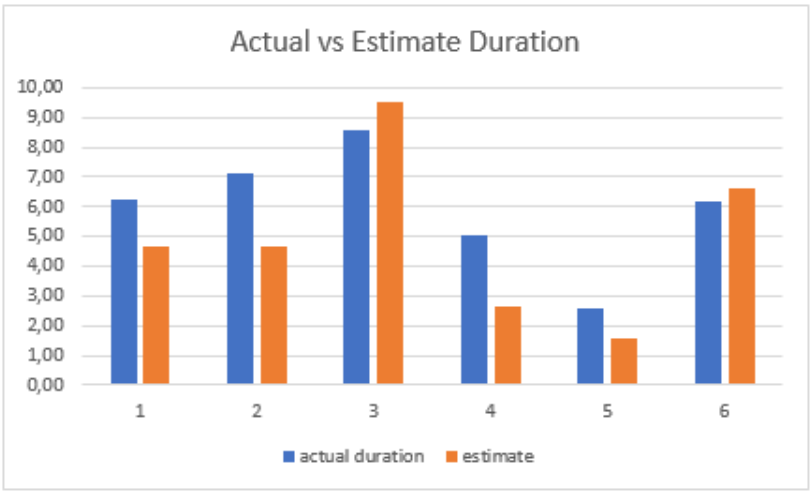

Figure 18. Comparing actual route times to estimate route times.

	actual duration	estimate	diff
1	6.27	4.67	1.60
2	7.13	4.80	2.33
3	8.58	9.50	-0.92
4	5.02	2.63	2.39
5	2.55	1.57	0.98
6	6.18	6.63	-0.45

Table 7. Illustrated data..

5.3 Route recommendation

The advantage of the timeslot-based approach is that it can capture and consider dynamics in the road traffic at a fine granularity. Compared to a more macroscopic data analysis over a longer time period, such a fine-grained approach may discover routes that may be generally busy, but at specific time intervals may still be rather empty. Conversely, it can avoid roads that generally seem unloaded, but at specific periods may be congested.

To assess the traffic dynamics that might be present in our dataset, which could yield improvement opportunities for the timeslot-based approach, we compute the difference in speed for the same edges over consecutive timeslots. This is done only for edges whose speed was calculated based on data contributed from at least 5 different drivers.

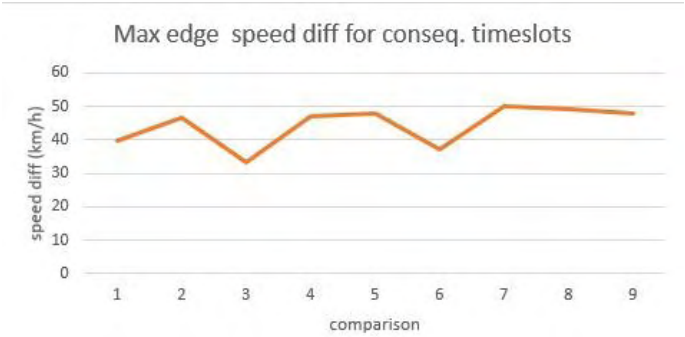

Figure 19. Comparing actual route times to estimate route times.

Figure 19 shows the maximum per-edge difference recorded over all edges between consecutive timeslots. The time period covered by the dataset is 50 minutes, so there are 10 timeslots in total, comparison 1 compares timeslot 1 to timeslot 2, etc. We observe that there are significant differences. Given that the timeslots we use are relatively small, some extreme differences are most likely due to traffic lights, and thus may not reflect the actual traffic conditions on the road. Still, these differences also indicate that timeslot-based routing has some potential.

Next, to see how such differences may affect routing decisions, we perform an experiment as follows. First, we pick a number of trips between different popular starting and destination points, with their start times and length chosen so that every trip spans across several timeslots. Then, for each such trip, we compute the best route using our approach with a different profile for each timeslot as well as using the single profile.

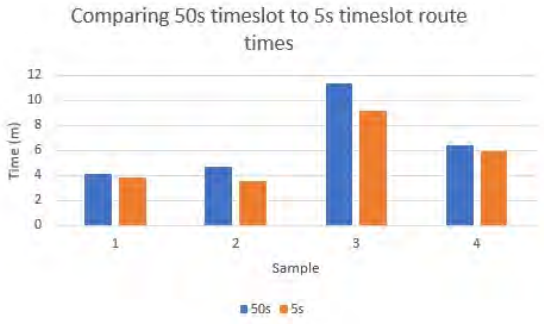

Figure 19. Comparing actual route times to estimate route times.

The results are shown in Figure 20. It can be seen in many cases the timeslot-based approach produces faster routes. However, there are also cases where the single (macroscopic) profile yields a better result.

Admittedly, the data set we used in our experiments is too narrow: it spans just 55 minutes of a single day of the week. Consequently, the above results should be taken with a grain of salt, and a much more extensive dataset is needed in order to be able to draw more concrete conclusions, as well as to evaluate the approach for its ability to predict traffic situations in the future (e.g., for the same or similar days of the week). Nevertheless, we believe that our results show the potential of the timeslot approach.

Chapter 6: Related Work

Similar work includes T-Drive [8], a platform treating taxi drivers as city road experts attempting to provide routing services based on taxi trajectories based on data collected by over 33.000 taxis in a period of 3 months. While our approach treats taxis as sensing devices collecting information about the traffic in roads they operate in, T-Drive uses taxi trajectories to construct preferred routes. T-Drives main challenge was its low sampling rate of 2-5 minutes per point, this caused lack of information regarding the respective routes each taxi followed and prediction models had to be used. Even though our sample data was produced by a much smaller number of taxis, our sampling rate of 6 seconds is significantly higher resulting in more accurate routing paths.

An attempt to construct a model of traffic density based on large scale taxi traces is presented in [9]. Instead of measuring traffic as of vehicle speed values, traffic is measured as the number of vehicles present at each road at a given time. Like in our approach, historical data is used to predict future conditions. Again, the sampling rate was 1 minute which is significantly lower than our sampling rate of 6 seconds.

A study exploiting taxi traces to detect trends in pick up and drop off points and their correlation to the land usage of the area is discussed in [10]. Claiming that land usage plays a key role in traffic as busier areas with more drop offs and pickups would also mean more traffic in the road. This research attempts to detect source and sink areas in relation with land usage to generate suggestions for planning land usage in a batter way so as to avoid traffic hotspots.

Chapter 7: Conclusions and Future Work

The results from the simulation seem encouraging but not dependable for routing purposes at this point. Further processing of the data needs to be made in order to get the best out of the available information. The main issue for the future is finding ways to fill the gaps in the traffic data. Nevertheless, a rather robust system prototype is now in place, which can be extended either to specific change parts of the data processing pipeline or to add new data processing stages, so that new method can be easily tested and evaluated.

This work can be extended in various ways. Switching the profile data of OSRM for another timeslot is very slow, because all nodes need to be reprocessed from scratch. One possible solution is to modify OSRM (which is open source) to use proper database technology for storing the traffic values internally. One could also vary the size of timeslots, using fine-grained timeslots only during periods with high traffic dynamics, and much bigger timeslots to cover periods where traffic remains stable. This could greatly reduce the number of profiles that needs to be re-loaded into OSRM when computing a (time wise) long trip. Another improvement would be to augment the current implementation by adding a new routing mechanism. It would also be useful to develop methods for detecting client drop-off and pick-up events in order to drop the respective data points early on. This could significantly improve data quality downstream, even at much lower data sampling rates. For roads segments with very few or no samples for a given timeslot, one could re-use data from timeslots with similar global traffic patterns, extrapolate available data for previous/following linear road segments for the same timeslot, or try to inherit data from roads with similar characteristics. Finally, additional contextual parameters, like weather conditions and local/social events, can be considered to analyze their effect on the traffic and produce better route recommendations for time periods with similar characteristics.

References

[1] J. Ginsberg, M. Mohebbi, R. Patel, L. Brammer, M. Smolinski and L. Brilliant, "Detecting influenza epidemics using search engine query data", *Nature*, vol. 457, no. 7232, pp. 1012-1014, 2008.

[2] Stathopoulos, A. and Karlaftis, M. (2001). Temporal and Spatial Variations of Real-Time Traffic Data in Urban Areas. Transportation Research Record: Journal of the Transportation Research Board, 1768, pp.135-140.

[3] "OpenStreetMap", *OpenStreetMap*, 2017. [Online]. Available: http://www.openstreetmap.org. [Accessed: 07- Oct- 2017].

[4] "Project-OSRM/osrm-backend", *GitHub*, 2017. [Online]. Available: https://github.com/Project-OSRM/osrm-backend. [Accessed: 07- Oct- 2017].

[5] "Cite a Website - Cite This For Me", *Leafletjs.com*, 2017. [Online]. Available: http://leafletjs.com/. [Accessed: 07- Oct- 2017].

[6] "Leaflet Heatmap Layer Plugin", *Patrick-wied.at*, 2017. [Online]. Available: https://www.patrick-wied.at/static/heatmapjs/plugin-leaflet-layer.html. [Accessed: 07- Oct- 2017].

[7] "How can I quickly estimate the distance between two (latitude, longitude) points?", *Stackoverflow.com*, 2017. [Online]. Available: https://stackoverflow.com/questions/15736995/how-can-i-quickly-estimate-the-distance-between-two-latitude-longitude-points. [Accessed: 07- Oct- 2017].

[8] Jing Yuan, Yu Zheng, Xing Xie and Guangzhong Sun, "T-Drive: Enhancing Driving Directions with Taxi Drivers' Intelligence", *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 220-232, 2013.

[9] P. Castro, D. Zhang and S. Li, "Urban Traffic Modelling and Prediction Using Large Scale Taxi GPS Traces", *Lecture Notes in Computer Science*, pp. 57-72, 2012.

[10] Y. Liu, F. Wang, Y. Xiao and S. Gao, "Urban land uses and traffic 'source-sink areas': Evidence from GPS-enabled taxi data in Shanghai", *Landscape and Urban Planning*, vol. 106, no. 1, pp. 73-87, 2012.