# W-edge: WEIGHING THE EDGES OF THE ROAD NETWORK

Rade Stanojevic, Sofiane Abbar, Mohamed Mokbel
Qatar Computing Research Institute, HBKU, Doha, Qatar

## ABSTRACT

Understanding link travel times (LTT) has received significant attention in transportation and spatial computing literature but they often remain behind closed doors, primarily because the data used for capturing them is considered confidential. Consequently, free and open maps such as OpenStreetMap (OSM) or TIGER, while being remarkably accurate in capturing geometry and topology of the road network are oblivious to actual travel times. Without LTTs computing the optimal routes or estimated time of arrival is challenging and prone to substantial errors. In this work we set to enrich the underlying map information with LTT by using a most basic data about urban trajectories, which also becomes increasingly available for public use: set of origin/destination location/timestamp pairs. Our system, W-edge utilizes such basic trip information to calculate LTT to each individual road segment, effectively assigning a weight to individual edges of the underlying road network. We demonstrate that using appropriately trained edge weights, the errors in estimating travel times are up to 60% lower than the errors observed in OSRM or GraphHopper, two prominent OSM-based, traffic-oblivious, routing engines.

## CCS CONCEPTS

• **Information systems → Geographic information systems**;

## KEYWORDS

Link travel times, Ridge regression, Maps.

**ACM Reference format:**

In *Proceedings of 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, November 6–9, 2018 (SIGSPATIAL '18),* 5 pages.

## 1 INTRODUCTION

Given a road network represented as a set of nodes and *weighted* edges, a source node, and a destination node, the shortest path operation finds a path from the source node to the destination node that minimizes the sum of edge weights. Finding shortest (or quickest) paths is critical for a number of spatial queries such as navigation, route planning or isochrone calculation. The most common way to define edge weights is by link travel time (LTT): an *average* time a vehicle spends on the road between two neighboring nodes in the graph which defines the road network. We use terms LTT and edge weight interchangeably throughout this paper, but note that the methodology proposed here can be seamlessly applied to extract per-edge cost different to duration, such as $CO_2$ emissions or fuel consumption [11].

The actual travel time between any two points can be highly variable due to variations in traffic demand, traffic lights, uncertain arrivals at the intersections, traffic light dynamics, individual driving behavior, etc. Hence, obtaining such edge weights or LTTs is highly non-trivial and a substantial amount of literature has been devoted to derive such information. Unlike the case for edge length and maximum speed that are mostly publicly available either through governmental or public websites (e.g., OpenStreetMap (OSM) or TIGER), accurate edge weights inferred either through loop detectors [1], ANPR [5], private GPS traces [3] are considered as proprietary information and not accessible to public. This sets apart commercial routing engines (e.g. Google Maps) from those who use publicly available data, like OSRM [7] and Graphhopper [4], where LTTs are inferred using heuristics and available road metadata[1]. To verify this, we performed comprehensive experiments in traffic-oblivious, OSM-based, routing engines OSMR and Graphhopper and found that they exhibit very large errors, greater than 50% in peak hours, in estimating travel times; see Section 4. This demonstrates the need to have publicly accessible accurate edge weights, which will have great impact in developing freely accessible and *accurate* shortest path operations.

In this paper we describe a methodology, W-edge, for enriching the road network with LTTs which utilizes nothing more than origin/destination location and timestamp information. We focus on data in this form as it is the 'lowest common denominator' of the trajectory datasets available for public use. While a number of cities have already publicly shared such taxi trajectories, we expect many other urban areas to follow. In particular, Freedom of Information Act (FOIA) [2] is a powerful tool in the hands of public [10] to request and obtain data in this form from publicly regulated transport organizations, such as taxi or public buses.

## 2 PROBLEM FORMULATION

The road network is represented as a graph, where each node is a unique $(latitude, longitude)$ pair and an edge between two nodes exists if and only if there is a road directly connecting them. Nodes lying in the middle of two-way streets often have in-degree and out-degree equal to 2; nodes lying on the highways have in-degree and out-degree 1; nodes at the intersection of two roads would have either in-degree or out-degree greater than 1.

A journey $\tau$ is represented by the available information: $I_\tau = [(lat, lon)_{orig}, tstamp_{orig}, (lat, lon)_{dest}, tstamp_{dest}, mileage]$. With each journey we associate a path in the road network given by a list of edges in the graph representing the road network $P_\tau = [e_0, \ldots, e_l]$. Effectively we want to do the map-matching using only origin and destination of the trajectory. To that end we query OSRM, an OSM-based routing engine, and denote by $P_\tau$ the

[1]E.g. speed limit, number of lanes, road type, etc.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*SIGSPATIAL '18, November 6–9, 2018, Seattle, WA, USA*
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5889-7/18/11…$15.00
https://doi.org/10.1145/3274895.3274916

result of the shortest path query using the journey $\tau$ origin and destination [6]. Obviously, $P_\tau$ does not necessarily correspond to the actual route traveled and here the mileage information can be used to eliminate those journeys for which the length of $P_\tau$ does not match *mileage*. Namely, from the set of available journeys, we consider only those 'reliable' journeys that satisfy:

$$mileage(1-\epsilon) < Length(P_\tau) < mileage(1+\epsilon), \tag{1}$$

for a small $\epsilon$. In the rest of paper we use $\epsilon = 0.05$, which allows high level of confidence that actual journey traveled over the corresponding path, and keeps a substantial fraction (around 40% of the total) of reliable journeys for training the model.

We denote by $\Gamma$ the set of reliable journeys represented by $(I_\tau, P_\tau)$, for $\tau \in \Gamma$. We aim to find weights of the edges in the underlying road graph that minimize

$$\sum_{\tau\in\Gamma}\Big(\sum_{e\in P_\tau} W_e l_e - \delta_\tau\Big)^2, \tag{2}$$

where $l_e$ is the length, in meters, of the edge $e$. Note that for given journey $\tau$, $\sum_{e\in P_\tau} W_e l_e$ represents an estimate of travel time (in seconds), while $\delta_\tau$ represents the observed travel time. The above sum effectively measures the sum of squares of the difference between the estimated and actual travel times for the set of journeys $\Gamma$. In other words, we look for weighing of the edges of the road network such that for every journey which follows a path $P_\tau$ its duration is accurately approximated by the sum of the weights of the edges on the path $P_\tau$.

An important remark here is that weights which minimize the errors (2) do not necessarily correspond to effective configuration for computing the shortest paths. Namely, allowing negative edge weights, as in [12], can disturb shortest path computation by not allowing convergence. This issue has been recognized in [8] which constrains the weights to be non-negative. However, even with this constraint, the non-negative least square solver LSEC [8] may (and it does) converge to a solution which associates zero weight to a large fraction of edges which allows shortcuts in the shortest-path computation and create sub-optimal routes and inaccurate ETA estimates; see our Technical report [9] for more details. Hence, we propose an alternative way for computing edge weights which rely on Ridge regression.

# 3 W-EDGE: WEIGHING THE EDGES OF ROAD NETWORK

In this section we describe W-edge system which takes as input a graph representation of the road network and a cohort of basic journey information and outputs for each of the graph the expected time, in seconds, a vehicle needs to traverse that edge. We refer to that time as weight or link travel time. Computing the edge weight can be oblivious to time of the travel in which it would capture the 'average' travel time, but edge weight can also depend on the time of the day and/or day of the week in which case we would have weight which is not a scalar but rather a function of time.

As we discuss above, minimizing (2) blindly, without taking into account the physical constraints which limit the actual travel time over any given edge, may lead to weights which are ineffective for the most critical routing primitive: shortest path computation. Additionally, the underlying road network can be very large (e.g. graphs which represent the road network of the metropolitan areas we study have hundreds of thousands of edges) grouping the edges in a structured way can significantly reduce the dimensionality of the problem and allow scaling the solver to millions of trajectories. Finally, to avoid over-fitting the model we focus only on edges which contain a non-trivial fraction of trajectories.

At this point we assume that every journey is map matched using origin/destination pairs and that we have filtered all of the journeys which do not satisfy the mileage constraint, as described above.

We split W-edge into three sequential phases: (1) heavy edge detection (to avoid over-fitting when low-frequency edges are used in the model), (2) heavy road detection (for dimensionality reduction) and (3) constraint-aware linear regression (for ensuring that edge weights correspond to physical constraints).

## 3.1 Heavy edges inference

Many journeys share large fraction of their trajectory with other journeys, yet they may have a few edges (typically near the origin or destination) which may be shared with a few or none other trajectories. Regression problems of type (2) which allow each edge in the graph, independently of its 'popularity' to act as a regression feature can easily lead to over-fitting. For that reason we focus on edges which support a large number of trajectories, which we call *heavy edges*. For a training set of trajectories, the set of heavy edges $H$ is derived by sorting the edges according the number of trajectories which pass them and taking the top $h_l$ of them. Here, $h_l$ the number of heavy edges, is a configurable parameter which controls the complexity of the model on one hand, and the execution time as well as the accuracy on the other. Throughout the paper we fix $h_l = 10000$, which captures most major roads in the three cities we study, yet resulting models are not computationally excessive.

Now instead of minimizing the sum (2) we focus on

$$\sum_{\tau\in\Gamma}\Big(\sum_{e\in P_\tau\cap H} W_e l_e + W_0 \sum_{e\in P_\tau\setminus H} l_e - \delta_\tau\Big)^2. \tag{3}$$

where $W_0$ is a unique weight of the light edges which we assign to all non-heavy edges. Above process reduces the complexity of the model (number of regression features) for 1-2 orders of magnitude in the 3 cities we study by effectively assigning the same weight to all light edges. The key insight here is that majority of shortest paths lie mostly on heavy edges, and therefore the weight of light edges has marginal effect on shortest paths or their length(duration).

## 3.2 Heavy road detection

Many edges appear in the trajectories simultaneously. By ensuring that they have the same weight we can substantially reduce the dimensionality of the problem. To that end we will group heavy edges together if they belong to the same trajectories. More formally, we split the set $H$ of heavy edges into subsets $H_1, \ldots, H_r$ such that:

$$(\forall i)(\forall \tau \in \Gamma)(\forall e_1, e_2 \in H_i) e_1 \in \tau \iff e_2 \in \tau.$$

In simple words if an edge lies on a trajectory, than all the other edges from its group must lie on that trajectory.

We refer to the (disjoint) sets $H_1, \ldots, H_r$ as heavy roads, as they typically group neighboring edges which form a road or part of a road.

Using the heavy road nomenclature we will rewrite the (3) as:

$$\sum_{\tau \in \Gamma} \Big( \sum_{g: P_\tau \cap H_g \neq \emptyset} W_g L_g + W_0 \sum_{e \in P_\tau \setminus H} l_e - \delta_\tau \Big)^2. \quad (4)$$

where $L_g$ is the length of the heavy road $H_g$:

$$L_g = \sum_{e \in H_g} l_e$$

Thus the number of unknowns in (4) is $r + 1$, where $r$ is the number of heavy roads: one weight for each of $r$ heavy roads and one weight for all other 'light' edges. In the data from three cities we study, the number of heavy roads $r$ is 3-4 times smaller than the number of heavy edges.

### 3.3 Enforcing physical constraints via Ridge regression

Authors of [8, 12] solve (2) allowing negative or zero weights which can fundamentally harm the shortest path computation. Here we strive to not only solve the appropriate numerical regression problem but also keep weights physically realistic. To that end, note that weight $W_g$ is inverse of the speed on the relevant road segment and is measured in $sec/m$. However, on each road segment the speed is limited and that information is often captured by the map itself; e.g. in OSM using a tag *maxspeed* (in $km/h$). Denoting $maxspeed_g$ the speed limit at road segment $g$, the physical constraint (accounting for appropriate unit change from $km/h$ to $sec/m$) for weights becomes:

$$W_g \geq 3.6/maxspeed_g \quad (5)$$

Note that we do not enforce upper limit on weights, as certain roads may become extremely congested (in particular periods of the day/week) and we would like to capture and preserve that information via high edge weight. To ensure that constraint (5) holds for all $g$ we propose to use Ridge regression regularization. Namely we add regularization term to (4) that penalizes weights which largely deviate from the average speed:

$$\sum_{\tau \in \Gamma} \Big( \sum_{g: P_\tau \cap H_g \neq \emptyset} W_g L_g + W_0 \sum_{e \in P_\tau \setminus H} l_e - \delta_\tau \Big)^2 + \alpha \sum_g (W_g - \sigma)^2. \quad (6)$$

Here $\sigma$ is the inverse of the average speed observed by all the journeys in our data. Parameter $\alpha$, represent the regularization strength. A small $\alpha$ allows large variability between $W_g$'s, a lot of violations of physical constraint (5) and can potentially lead to over-fitting. A very large $\alpha$ may put too much emphasis on the regularization term neglecting errors we strive to minimize. We choose the hyper-parameter $\alpha$ using a grid search over a small validation set we withdraw from the training cohort.

We use scikit-learn Python library to find $W$'s which minimize (6) and can scale to millions of trajectories, thanks to dimensionality reduction described above and sparse matrix representation of the feature matrix.

With an appropriate regularization strength $\alpha$ (see our Tech report [9] for details on how we automatically tune $\alpha$), Ridge regression regularization indeed forces most of $W_g$'s to be close to $\sigma$ and we empirically observe that a large majority of segment weights satisfy the constraint (5).

The weights derived by minimizing (6) satisfy the speed limits in over 99% of cases. For a small minority of edges which violate the speed limit constraint we hard code the weights to be exactly equal to $3.6/maxspeed_g$. Thus weight $\bar{W}_g$ at road segment $g$ is

$$\bar{W}_g = max(3.6/maxspeed_g, W_g),$$

where $W_g$ are obtained by minimizing (6). This last step, has a minor effect on the overall ETA estimation, measured through cost (4) since it applies to a minor fraction of the road network, but it eliminates artificial shortcuts which may arise by allowing edges with very high speeds, say over $120kmph$.

## 4 EVALUATION

### 4.1 Data

For the purpose of evaluating `W-edge` we take advantage of the data generated by a 4000-car taxi fleet in Doha from February 2018, with 1.6M journeys which in average last 13.9 minute. The underlying graph has around 175K nodes and 320K edges. In the Tech report [9] we examine in depth the behavior of `W-edge` using the data from New York and Porto taxi fleets in addition to Doha.

As we explain above, the map-matching using only O-D pairs is done using OSRM and consider only those journeys whose mileage matches the OSRM route within 5% threshold.

### 4.2 `W-edge` vs. traffic-oblivious shortest paths

Our first question is: **How does `W-edge` compare with traffic oblivious OSM-based routing engines in terms of ETA errors?** To that end we choose the two default engines that OSM landing page[2] offers: Open Source Routing Machine (OSRM) [7] and Graphhopper (GH) [4]. Both of those routing engines utilize the underlying OSM data to build the road network graph equipped with weights which are used for computing the optimal routes. The weights OSRM and GH use are derived from the OSM metadata in a way to prioritize routing over motorways and other primary roads. For each (origin,destination) pair both OSRM and GH return a route together with ETA (estimated time of arrival, or route duration) which is derived by solving an appropriate quickest-path query. While both OSRM and GH have a web-based API, OSRM also offers code to set-up a local-server on own premises, which we do, in order to submit a high volume of queries to it. Namely demo OSRM and GH servers with no optimization allow us throughput of around 1 query per second while the local copy of OSRM allows us 100 queries per second.

For each city we first filter out all the trajectories which do not match the length of OSRM path and are left with about 40% of the original trajectories. We split these trajectories into training (80%) and test data (20%). And we also withdraw a small fraction (5%) of the training data for validation and tuning the hyper-parameter $\alpha$.

In the training phase we use the training data (minus validation set) to infer the weights of individual edges as detailed in Section 3.

[2] https://www.openstreetmap.org/directions

Rade Stanojevic, Sofiane Abbar, Mohamed Mokbel
Qatar Computing Research Institute, HBKU, Doha, Qatar

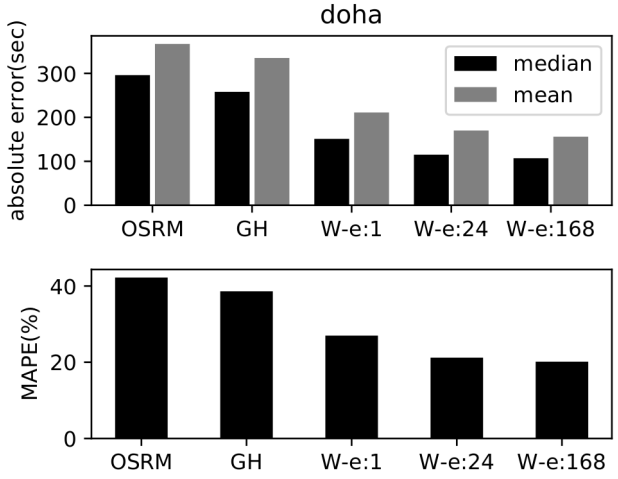


**Figure 1: Mean/median absolute error and mean average precision error (MAPE). OSRM and GH have substantial errors. Using W-edge reduces all three metrics. Adding temporal granularity in how frequently weights may change additionally reduces errors. E.g. Median absolute errors in Doha using OSRM or GraphHopper are 296 and 258 seconds, respectively. Using W-edge:168 they go down to 107 seconds, a reduction of 64% and 59%, to OSRM and GH, respectively.**

In order to differentiate among different levels of temporal granularity we train three different models: W-edge:1 single weight per edge; W-edge:24 24 weights per edge, corresponding to hour of the day and W-edge:168 168 weights, for each hour of the day for each day of the week

For each journey $\tau$ in the test data we compare the actual travel time with 5 values: OSRM journey duration, Graphhopper journey duration as well as the sum of weight edges over the route $P_\tau$, using W-edge:1, W-edge:24 and W-edge:168. We evaluate three different metrics: median absolute error (in seconds), mean absolute error (in seconds) and mean absolute percentage error (MAPE, in %) and report the results in Figure 1.

From the figure we can learn several lessons. First, while querying traffic-oblivious routing engines may offer reasonable routes the actual ETA is very inaccurate. The mean absolute error is between 6 and 7 minutes in both OSRM and GH, which is very large, especially given that average trip duration is only 13.9 minutes. Second, in all three metrics GH routing engine outperforms OSRM. Third, adding a single traffic-aware weight per edge can substantially reduce the ETA errors compared to traffic oblivious OSRM and GH. Fourth, W-edge:24 offers substantial reduction in errors compared to W-edge:1, while differentiating between different days of the week in W-edge:168 does offer improvement to W-edge:24, albeit it is quantitatively relatively small.

## 4.3 When do errors appear?

Throughout the day there is a substantial variability of traffic conditions and we examine how such variability affects the accuracy of traffic-oblivious OSRM or GH as well as traffic-aware W-edge. Similarly to the previous section we evaluate the errors on the test journeys using the three different W-edge models, as described above. We slice the day in 24 hour-long periods and within each hour we evaluate the errors between the actual and the predicted travel time. In Figure 2 we report the mean absolute error across the day; the results for median absolute error and MAPE are qualitatively similar and we omit them here.

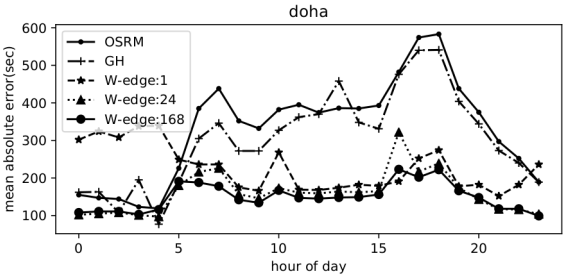


**Figure 2: Mean absolute errors throughout the day**

From Figure 2 we can observe that in all studied scenarios the errors over night are generally lower than the errors during the day. Mean absolute errors (MAE) of traffic-oblivious OSRM and GH can be as large as 10 minutes in the peak hour, while traffic-aware W-edge keeps MAE less than 3.5 minutes throughout the day. Note that majority of journeys happen during the day and hence W-edge:1 optimizes for the errors in that period and hence has substantial errors over night. Interestingly enough, both OSRM and Graphhopper have fairly low errors over night, indicating that the heuristics used by both engines to match the OSM metadata to travel times are suitable for off-peak periods, and consequently lead to large errors during the peak hours.

## 4.4 Public LTT data

As we said above, our primary motivation was enriching the public maps with link travel time information which could be freely used in a range of location based services. To that end, we publish the individual edge weights derived by W-edge and they can be found at: http://ds.qcri.org/people/rstanojevic/wedge/readme.txt.

## REFERENCES

[1] Benjamin Coifman. 2002. Estimating travel times and vehicle trajectories on freeways using dual loop detectors. *Transportation Research Part A: Policy and Practice* 36, 4 (2002).
[2] H. N. Foerstel. 1999. *Freedom of information and the right to know: The origins and applications of the Freedom of Information Act.* Greenwood Press.
[3] T. Hunter and et al. 2009. Path and travel time inference from GPS probe vehicle data. *NIPS Analyzing Networks and Learning with Graphs* (2009).
[4] P. Karich and S Schröder. Graphhopper. In *http://www.graphhopper.com.*
[5] E. Kazagli and H. Koutsopoulos. Arterial travel time estimation from automatic number plate recognition data. In *Proceedings of Annual TRB Meeting 2013.*
[6] Y. et al. Lou. Map-matching for low-sampling-rate GPS trajectories. In *Proceedings ACM SIGSPATIAL 2009.*
[7] D. Luxen and C. Vetter. Real-time routing with OpenStreetMap data. In *Proceedings ACM SIGSPATIAL 2011.*
[8] S. Mridha, N. Ganguly, and S. Bhattacharya. Link Travel Time Prediction from Large Scale Endpoint Data. In *Proceedings ACM SIGSPATIAL 2017.*
[9] R. Stanojevic, S. Abbar, and M. Mokbel. 2018. W-edge: Weighing the Edges of the Road Network. Technical Report, https://ds.qcri.org/people/rstanojevic/wedge_TechReport.pdf. (2018).
[10] Chris Whong. FOILing NYCs taxi trip data. 2014.
[11] B. Yang and et al. Using incomplete information for complete weight annotation of road networks. In *IEEE Transactions on Knowledge and Data Engineering, 2014.*
[12] F. Zheng H. Van Zuylen. Urban link travel time estimation based on sparse probe vehicle data. In *Transportation Research Part C: Emerging Technologies 2013.*