MapReuse: Recycling routing API queries

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

***Abstract*—Commercial maps often offer traffic awareness which is critical for many location based services. On the other hand free and open map services (such as government maps or OSM) are traffic oblivious and hence are of limited value for such services. In this paper we show that coarse information available from a commercial map routing API, can be dissected into fine-grained per-road-segment traffic information which can be reused in any application requiring traffic-awareness. Our system MapReuse queries a commercial map for a (relatively small) number of routes, and uses the returned routes and expected travel times, to infer travel time on each individual edge of the road network. Such fine-grained travel time information can be used not only to infer travel time on any given route but also to compute complex spatial queries (such as traffic-aware isochrone map) for free. We test our system on four representative metropolitan areas: Bogota, Doha, NYC and Rome, and report very encouraging results. Namely, we observe the median and mean percentage errors of MapReuse, measured against the travel times reported by the commercial map, to be in the range of 4% to 8%, implying that MapReuse is capable to accurately reconstruct the traffic conditions in all four studied cities.**

I. Introduction

Traffic information is critical in many location based services, and plays an important part in the success of many commercial map engines such as Google maps, Apple maps, Mapbox, HERE maps, Bing maps. Collecting traffic data is highly challenging and commercial map services consider such information as an important confidential asset. While fine-grain traffic information is not shared with public, commercial maps often allow to be queried for routing information, in which case a coarse information in the form of total travel time (or total average speed) is reported for a given path.

On the other hand, free and open map services such as OSM [11] or TIGER [4], while having a highly accurate geometric representation of the road network, are traffic-oblivious and have limited value for services which require traffic-awareness; see Section IV where we examine the errors in estimating travel time using open maps with no traffic awareness.

In this work our aim is to exploit the results of a limited number of queries to commercial map routing API (in the form of coarse path-level information about the traffic), to construct per-road-segment traffic image of the city which in turn can be incorporated into any open map and used for arbitrary spatial queries requiring traffic awareness.

Our system is built to assist researchers and practitioners who need fine-grained traffic information which is often not available for public use.

A. Motivation

Our work is motivated by a countless array of location-based services which require traffic information. These include optimized route scheduling and route planning, location-aware recommendations, transportation modeling and planning.

An ecosystem of commercial maps has emerged to assist these services in optimizing their performance with traffic-awareness. For example a fleet of courier vans may want to schedule its routes to deliver a given set of parcels to simultaneously minimize fuel consumption and travel time. With dynamic traffic conditions, which vary throughout the day, solving such multi-dimensional optimization problem heavily relies on the actual traffic conditions.

Acquiring such traffic information is highly challenging. While some cities (mostly in the tech-hubs) collect and publicly share traffic information on most prominent road segments (e.g. motorways/freeways), it is the case that in a large majority of cities no such information is publicly available. Hence, organizations and services which require traffic information mostly, if not exclusively, rely on commercial map services which acquire traffic information primarily using millions of connected devices such as smartphones or personal navigation devices.

The target user of our system, coined MapReuse, falls into one of the following two categories: (1) *High-volume routing API applications*. For any application which consistently requests routes/travel times, in the order of magnitude of thousands to millions of calls per day, commercial map services will incur a substantial bill. We see MapReuse as a cost-efficient alternative which can allow such applications to reuse the relevant information in a flexible and accurate manner. (2) *Non-standard spatial queries*. Commercial maps normally have a short list of standard queries which are well documented and supported. More complex spatial queries are often unavailable, and would require substantial engineering effort and/or nontrivial cost to derive the results composing an array of standard queries available in the traffic aware map engine. One example of such complex spatial queries is traffic-aware Isochrone computation[1] [25]. Another example is computation of many-to-many (say more than 100 or even more than a 1000 sources and destinations) distance matrix critical for many contemporary traffic modeling applications [20] which could be obtained either by querying the commercial map engine for a nontrivial cost[2] or alternatively through low-accuracy traffic-oblivious heuristics such as OSRM [15]. Using MapReuse can gracefully introduce traffic awareness (and

[1]For a given point, and given travel time δ, isochrone line is defined as a set of points at distance δ from the point.

[2]The cost of obtaining such distance matrix grows quadratically with the size of the matrix. With current Google Maps pricing, obtaining a 1000×1000 distance matrix for one single timeslot would cost 10 thousand US dollars.

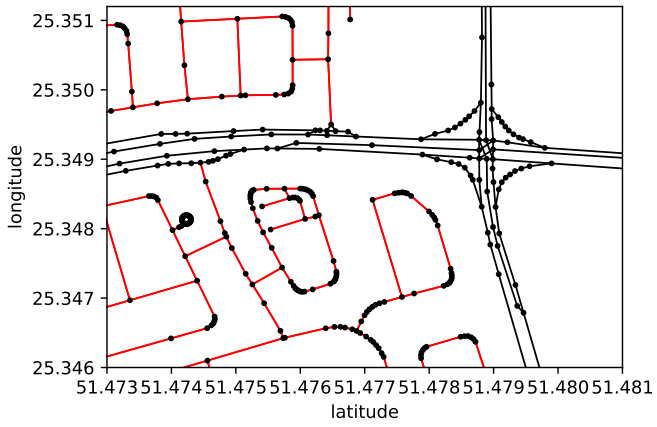

Fig. 1. Graph representation of a neighborhood in Doha. Black edges: one way streets; red edges: two-way streets

keep the cost low) to such complex and non-standard spatial queries which are currently not supported by the commercial map engines.

B. Our contributions

The main contributions of this paper are the following:

- We build the system, `MapReuse`, which consumes coarse total travel time information returned by querying a commercial map directions API and returns travel time of each individual edge of the graph representing the road network of the given city.
- We empirically evaluate the ability of `MapReuse`, to infer travel times on arbitrary routes, in four representative cities, and demonstrate that the mean and median percentage error between reported travel time and the `MapReuse` predictions is between 4% and 8% in all four studied cities: an order of magnitude reduction compared to the errors observed in OSRM, a prominent OSM-based routing engine.

By reconstructing traffic-aware per-road-segment travel time, `MapReuse` can substantially reduce the cost to the users which requires high-volume direction API queries (by reusing a fraction of the API queries), and at the same time enable a range of spatial queries currently not offered by the commercial map engines.

II. Problem formulation

As we explain in Section I, our focus is on inferring link travel times by querying commercial map directions API. Various commercial maps provide directions APIs which could be called by passing several parameters and return results in the form of a route. While the actual details vary, in the most generic form, user can call the directions API by providing origin and destination location (in the form of latitude/longitude pairs) along with departure time. The API returns one or more 'optimal' paths, together with their travel times, from the origin to the destination departing at the provided time.

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. See Figure 1 for an illustration of the graph in a neighborhood of Doha. While we work with OSM graph in this paper, inferring link travel time in any other road network base map using `MapReuse` would be straightforward.

A journey τ is represented by the available information: $I_\tau = [path_\tau, \delta_\tau]$. Here $path_\tau$ is represented as a list of (lat, lon) pairs returned by the API, and can be either dense (representing accurate geometry of the route) or sparse (with only important descriptors, such as turns at the intersections). Variable δ_τ represents the travel time over the $path_\tau$, estimated by the commercial map engine, for the journeys starting at the provided departure time.

We use OSRM [15] to map match the path of the journey τ to the road network, and denote by $P_\tau = [e_0, \ldots, e_l]$ the list of edges of the road network matched to the $path_\tau$.

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

$$\sum_{\tau \in \Gamma} \Big(\sum_{e \in P_\tau} W_e l_e - \delta_\tau \Big)^2, \tag{1}$$

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

An important remark here is that weights which minimize (1) do not necessarily correspond to effective configuration for computing the shortest paths. Namely, allowing negative edge weights, as in [32], can disturb shortest path computation by not allowing convergence. This issue has been recognized in [16] which constrains the weights to be non-negative. However, even with this constraint, the non-negative least square solver LSEC [16] 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 [22] for more details. Hence, we propose an alternative way for computing edge weights which relies on Ridge regression.

Before we proceed we would like to mention that our poster paper [23] reports many of the ideas behind `MapReuse`, although [23] focuses in annotating weights of the road

TABLE I
SYMBOL MAP.

Symbol	meaning
τ	journey
e	edge in road network
P_τ	path of journey τ - list of edges
ϵ	mileage matching threshold
Γ	set of training journeys
W_e	weight of edge e (in sec/m)
l_e	length of edge e (in m)
δ_τ	duration of journey (in sec)
H	set of heavy edges
h_l	parameter: number of heavy edges
W_0	weight of **all** light edges
r	number of heavy roads
H_g	heavy roads, $g \in \{1, \ldots, r\}$
W_g	weight of heavy road g
$maxspeed_g$	speed limit on road g (in km/h)
σ	inverse of average speed (in sec/m)
α	Ridge regularization strength

network using only origin-destination location/timestamp pairs sometimes publicly available from the taxi traces. In contrast MapReuse strives to infer the weights using the commercial map engine routing API calls which can be queried in virtually every city on earth for a fee.

III. MAPREUSE: WEIGHING THE EDGES OF ROAD NETWORK

In this section we describe MapReuse system which takes as input a graph representation of the road network and a cohort of basic journey information and outputs for each edge 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 (1) 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 four metropolitan areas we study in this paper - Bogota, Doha, NYC and Rome - 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 a large number 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 onto the OSM using the route returned by the API, as described in Section II.

We split MapReuse 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).

A. Heavy edges inference

Many journeys share large fraction of their trajectory with other journeys, yet they may have some edges (typically near the origin or destination) which may be shared with few or none other trajectories. Regression problems of type (1) 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 that 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 to the number of trajectories passing 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 four cities we study, yet resulting models are not computationally excessive.

Now instead of minimizing the sum (1) we focus on

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

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 four 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).

B. 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 (2) as:

$$\sum_{\tau \in \Gamma} \Big(\sum_{g : P_\tau \cap H_g \neq \emptyset} W_g L_g + W_0 \sum_{e \in P_\tau \setminus H} l_e - \delta_\tau \Big)^2. \qquad (3)$$

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 (3) 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 four cities we study, the number of heavy roads r is 3-4 times smaller than the number of heavy edges.

C. Enforcing physical constraints via Ridge regression

Authors of [16], [32] solve (1) 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 (also known as posted speed) at road segment g, the physical constraint (accounting for appropriate unit change from km/h to sec/m) for weights becomes:

$$W_g \geq 3.6/maxspeed_g \tag{4}$$

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 (4) holds for all g we propose to use Ridge regression regularization. Namely we add regularization term to (3) that penalizes weights which largely deviate from the average speed:

$$\sum_{\tau\in\Gamma}\Big(\sum_{g:P_\tau\cap H_g\neq\emptyset} W_g L_g + W_0 \sum_{e\in P_\tau\setminus H} l_e - \delta_\tau\Big)^2 + \alpha\sum_g (W_g-\sigma)^2. \tag{5}$$

Here σ is the inverse of the average speed observed by all the journeys in our data. Parameter α, represent the regularization strength. A small α allows large variability between W_g's, a lot of violations of physical constraint (4) and can potentially lead to over-fitting. A very large α may put too much emphasis on the regularization term neglecting errors we strive to minimize. We choose the hyper-parameter α using a grid search over a small validation set we withdraw from the training cohort; see below.

We use Scikit-learn Python library [19] to find W's which minimize (5) and can scale to a large number of trajectories, thanks to dimensionality reduction described above and sparse matrix representation of the feature matrix.

With an appropriate regularization strength α (see below for details on how we choose α), Ridge regression regularization indeed forces most of W_g's to be close to σ and we empirically observe that a large majority of segment weights satisfy the constraint (4), see Section IV-E for more details. 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

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

TABLE II
BASIC INFO ABOUT THE DATA

City	# Paths	avg duration	# nodes	# edges
Bogota	9674	18.2 min	228K	363K
Doha	5601	13.5 min	175K	321K
NYC	10649	18.1 min	299K	596K
Rome	8760	15.2 min	269K	310K

where W_g are obtained by minimizing (5). This last step, has a minor effect on the overall travel time estimation, measured through cost (3) 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 $120kmph$.

1) Choosing regularization strength α: We do not require α to be manually set, but rather automatically tune it. Our search space is $\alpha \in \{2^k;\ k=0,1,\dots\}$. We start by choosing $\alpha = 1$, obtain the weights $\bar{W}_g(\alpha)$ and measure the cost on the validation dataset (a small set withdrawn from the training cohort) by substituting those weights in (3) which we denote by $C(\alpha)$. We keep doubling α until $C(\alpha) > C(2\alpha)$, and choose α to be that terminal value, which effectively minimizes the cost $C(\alpha)$ in the above search space.

2) Number of weights per segment: If our goal is to capture traffic conditions averaged across long time periods, having a single weight per segment would provide reasonably good estimate of the time spent on any given road segment. However it is well known that traffic conditions exhibit periodic behavior on various time-scales: daily, weekly, yearly. For the applications which would require more accurate estimates of the traffic conditions at any given time one may want to train multiple models for capturing the traffic variability in time. For example we can derive one weight per edge for every hour of the day; or one weight per edge for every hour of the week (168 weights) to differentiate between weekdays and weekends. The key point here is to train on a subset of trajectories that fall into the appropriate time-slots, and as we shall see in the following section, having time-varying weights can substantially reduce the errors in estimating the travel time.

IV. EVALUATION

A. Data

As explained earlier, our system consumes data which are returned by querying commercial maps. In the present paper we utilize Google Maps directions API, but note that any other commercial map engine, which has directions API such as Bing, HERE, MapBox, could be dissected in virtually identical manner.

We study four prominent cities, from four different continents: Bogota, Doha, New York and Rome. For each city, we randomly select 5000 origin-destination pairs and for each pair we query Google Maps directions API four times, requesting the routes which connect the origin to the destination at four different times midnight, 6am, noon and 6pm on 16th of January 2019. We choose 16th of January as a workday in

the middle of the week (Wednesday) which is far enough in the future not to be affected by transient effects such as short-term congestion or road-closures.

For each triplet $(orig, dest, timestamp)$ the actual API call we submit is the following:
https://maps.googleapis.com/maps/api/directions/json?origin=orig\&destination=dest\&departure_time=timestamp\&key=our_key\&traffic_model=best_guess\&alternatives=true

Parameter $alternatives$ set to $true$ indicates that more than one (but up to three) routes are returned for each query. Traffic model parameter is set to the default $best\_guess$. To quote Google Maps Direction API guideline: "$traffic\_model = best\_guess$ (default) indicates that the returned $duration\_in\_traffic$ should be the best estimate of travel time given what is known about both historical traffic conditions and live traffic. Live traffic becomes more important the closer the $departure\_time$ is to now". Given that we query for traffic conditions several weeks into the future, the results are virtually independent of the live traffic.

Remark: OD pairs selection. As we mention above the OD location pairs are drawn randomly from the set of OSM nodes. We iterate through a list of randomly selected pairs (O, D) of OSM 2-way road nodes and accept them with a probability $min(1, (5km/d(O, D))^2$. By selecting nodes which lie on 2-way roads we eliminate the effects of inappropriate geocoding which matches a location to a road on the opposite side of the street (and may require a large detour for routing). The choice to prefer closer pairs to those far apart, is motivated by the observation that routes which connect far-apart locations almost exclusively utilize motorways and primary roads and hence carry little information about the traffic conditions on the secondary and tertiary roads. It would be interesting to understand the optimality of OD pair selection for capturing most traffic information with as little as possible API calls, though such analysis is out of scope of the present paper.

For each $(orig, dest, timestamp)$ triplet, the API call returns a list of up to three routes together with the estimated travel time over that route at the given departure time. We use OSRM map matching service to match each route to the OSM. In Table II we report the total number of map-matched routes in the 4 cities using 5000 API calls, at midnight timeslot (for other three timeslots the numbers are similar). For Bogota, NYC and Rome we get around 2 routes per OD pair, in average, while in Doha we get only around 1 map-matched route per OD pair in average. We believe that poor map-matching success in Doha is due to a very large number of road closures and upgrades which results in many inconsistencies between Google Maps and OSM.

For each city and each of the four time slots, we use the routes obtained from 80% of the API calls for training the model, and the remaining 20% of the routes for testing.

B. `MapReuse` vs. traffic-oblivious routing engines

Our first question is: **How does `MapReuse` compare with traffic oblivious OSM-based routing engine in terms of ETA errors?** To that end, we choose the default engine that OSM landing page[3] offers: Open Source Routing Machine (OSRM) [15]. OSRM utilizes the underlying OSM data to build the road network graph equipped with weights which are used for computing the optimal routes. The weights OSRM uses are derived from the OSM metadata in a way to prioritize routing over motorways and other primary roads. For each (origin,destination) pair OSRM returns a route together with ETA (estimated time of arrival, or route duration) which is derived by solving an appropriate quickest-path query. While OSRM has a web-based API, it also offers code to set-up a local-server on own premises, which we do, in order to be able to maintain high-throughput. Namely web OSRM servers allows throughput of around 1 query per second while the local instance of OSRM gets us 100 queries per second.

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

We look at how closely OSRM and `MapReuse` approximate the total travel time reported by the commercial map for the trajectories in the test cohort. For each route τ from the test cohort we consider the travel time reported by the commercial map, δ_τ as ground truth and compare it with two values: $\delta_\tau(OSRM)$: travel time reported by OSRM and δ_τ(`MapReuse`): travel time derived by `MapReuse`, defined as the sum of `MapReuse`-inferred travel times across all the edges traversed by τ, P_τ.

To compare how well OSRM and `MapReuse` approximate the travel time, we evaluate four different metrics: median and mean absolute error (in seconds), and median and mean percentage error (in %) and report the results in Figure 2.

From the figure we can learn several lessons. First, while querying traffic-oblivious routing engines may offer reasonable routes the actual ETA is highly inaccurate. With OSRM, in the cities of New York and Bogota the mean absolute error at 6pm (afternoon peak) is 15 and 11.7 minutes, which translates to over 40% of mean percentage error. Second, `MapReuse` offers an order of magnitude reduction in errors compared to bare-bones OSRM: the mean and median absolute errors are close to 1 minute, while the mean and median percentage errors are in the range of 4% to 8% in all four studied cities. This demonstrates the ability of `MapReuse` to capture the relevant traffic information on a per-road-segment level and use it to accurately capture the travel time on arbitrary path.

C. How many API calls?

A critical parameter which influences the quality of the inferred traffic information is the number of API calls we use to train the model of individual edge weights. On one hand, more API calls result in more accurate representation of the travel times throughout the city. On the other hand, more API calls result in a higher cost for data acquisition. In this paragraph we examine the trade-off between the number of API calls used in the training phase of `MapReuse` and the actual errors in travel times.

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

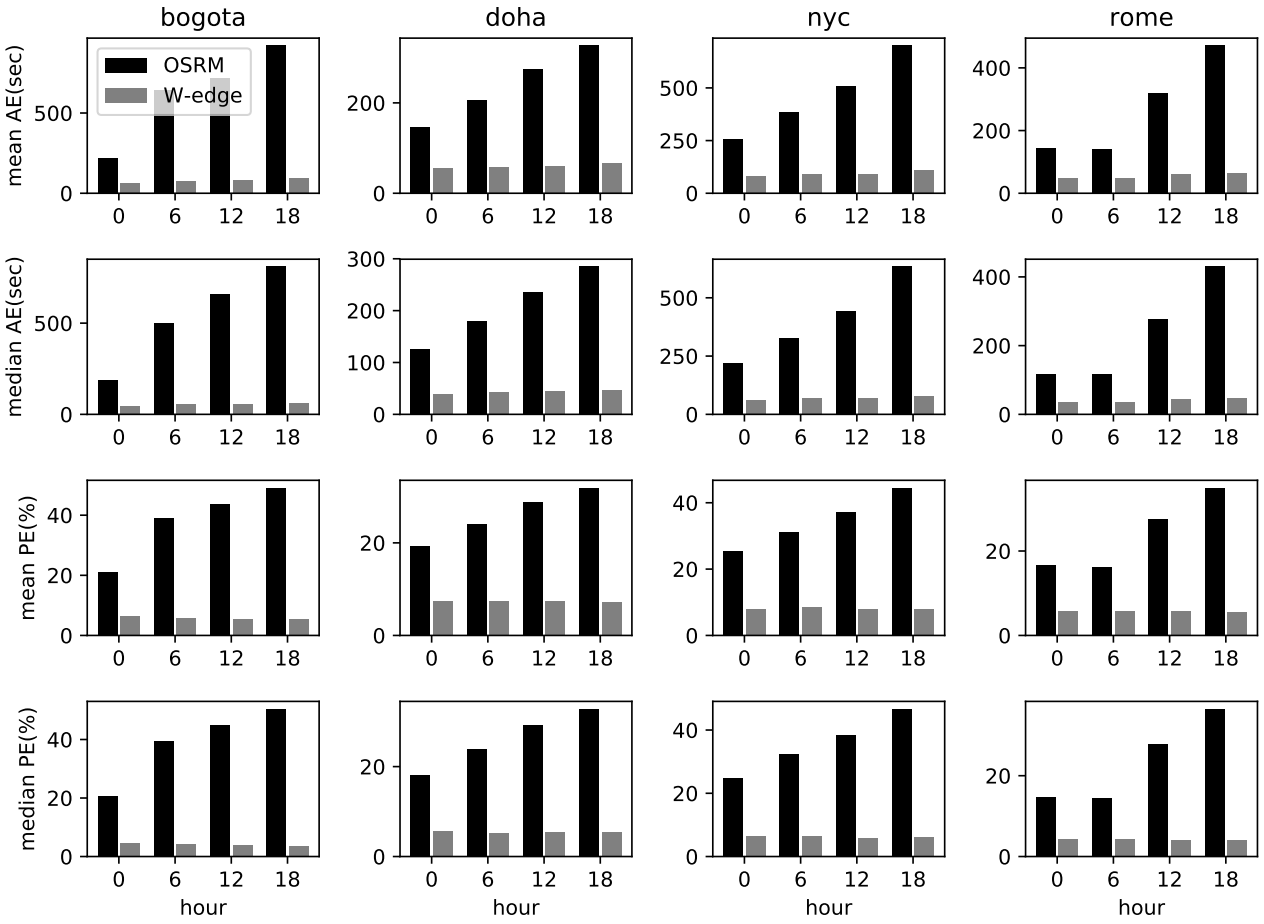

Fig. 2. Mean/median absolute error and mean/median average precision error in predicting the travel time reported by the commercial map (used as ground truth). OSRM has substantial errors. Using `MapReuse` reduces errors across all four metrics. E.g. Median absolute error in Bogota, during the 6pm time-slot, using OSRM is 810 seconds. Using `MapReuse` median absolute goes down to 61 seconds, a reduction of 92%, to OSRM. In Doha, NYC and Rome, the corresponding reduction in median absolute error is 84%, 88% and 89%, respectively.

To do that we limit the number of API calls used in the training phase of `MapReuse` to $n\_samples$ and evaluate the ETA errors on a set of 1000 paths not used in the training phase.

In Figure 3 we report mean and median absolute percentage errors (PE) using `MapReuse` at 6pm time slot for the four cities of interest and varying number of training journeys $n\_samples$ between 100 and 4000. Results in other 3 time-slots (midnight,6am and noon) are both quantitatively and qualitatively similar, and are omitted here. Intuitively, the larger the training data the smaller the errors. However we observe a law of diminishing returns here, with minor improvements with larger data. Namely, after 1000 API calls, the accuracy of the `MapReuse` model improves only marginally with more data. Note that training the `MapReuse` model even on a small dataset with several hundred of journeys would outperform traffic oblivious OSRM. However to get most of our model one needs several thousands of API calls per city per time-slot.

Remark. We would like to emphasize here that we strongly believe that the number of necessary API calls could be further reduced with a smarter selection of origin-destination pairs. With a better coordination on how we choose O-D pairs, we could avoid substantial redundancy which is a result of random O-D pair selection. However, such analysis is left for future work.

D. Heavy roads

A very important step in `MapReuse` is identification of heavy edges and heavy roads (which are groups of heavy edges). `MapReuse` effectively infers speed on each heavy road individually and simultaneously assigns a single weight (speed) to all the light roads. In Figure 4 we depict the road network of Bogota downtown emphasizing the heavy roads. We can observe that by focusing on heavy roads, we effectively capture most of the major road arteries, and eliminate low-volume roads which may create overfitting issues in the model training phase of `MapReuse`. However, not focusing on light roads means that certain traffic events (such as congestion or extraordinary fast light road) may not be captured by `MapReuse`. This is one of the factors which limits the accuracy of `MapReuse` weights, and can, to some extent,

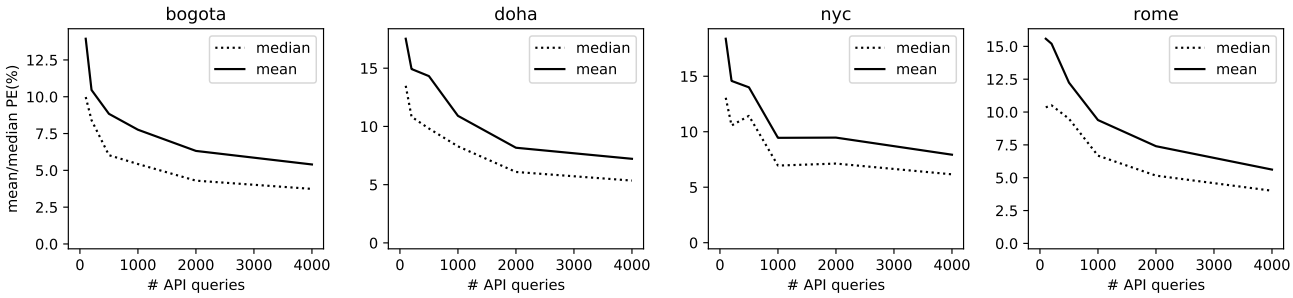

Fig. 3. `MapReuse` accuracy gracefully improves with more training journeys up to a point. From 1000 API calls, adding more data results in marginal improvements in accuracy.

explain the plateauing effect observed in errors (see Figure 3) where additional training data does not lead to noticeable improvements in the accuracy of `MapReuse`.

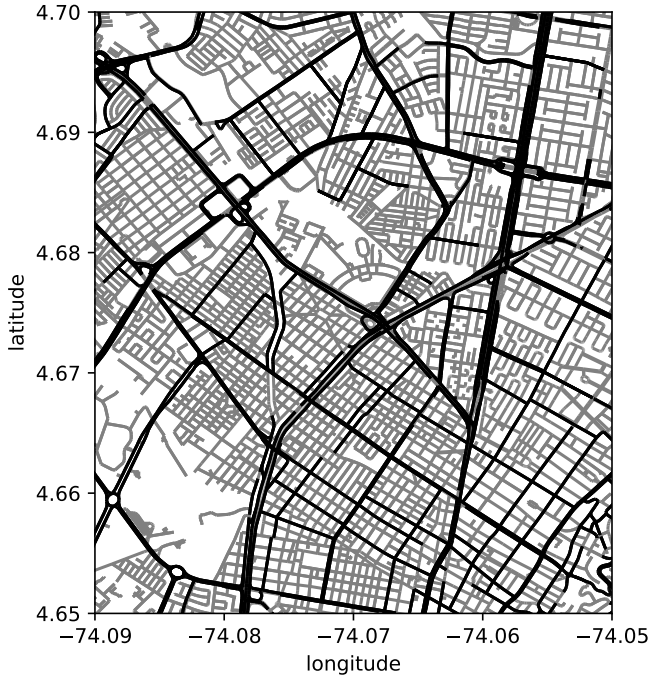

Fig. 4. Visual representation of Bogota downtown road network. Grey edges represent light roads, while black ones correspond to heavy roads.

E. `MapReuse` Relative speeds

In this paragraph we look into the question of how speeds inferred by `MapReuse` relate to the actual speed limits declared by Open Street Map[4]. In Figure 5 we depict the empiric CDF of the ratio between the inferred speed $3.6/W_g$, and the $maxspeed$ tag (both measured in km/h) for all road segments in the four cities we study here. As we can see from the figure, the weights derived by minimizing (5) satisfy the speed limits in a large majority of cases. While for the others, the weights are hard-coded to be exactly determined by the speed limit. The fraction of edges for which W_g violates the OSM-reported speed limit is less than 4% in all four cities.

[4]For those edges where no speed limit information is available in OSM we use the default speed limit in the corresponding country.

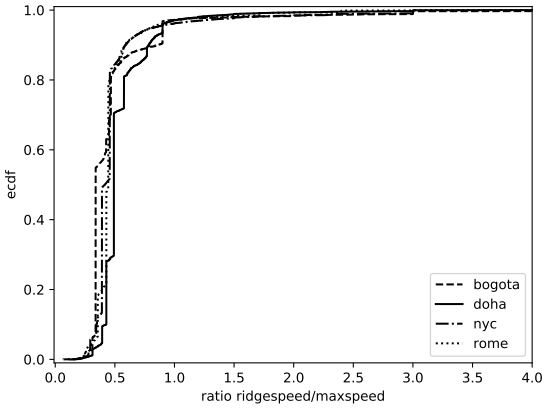

Fig. 5. Empiric CDF of the ratio $(W_g/3.6)/maxspeed_g$ for four cities we study, where W_g is obtained by minimizing (5). Ratio larger than 1 means that W_g allows travel over segment g faster than allowed $maxspeed_g$. In the four cities we study the fraction of such segments is $< 4\%$. Note also, that majority of the mass of the CDF is concentrated around 0.5, which indicates that for most segments average speed is around half of the maximum speed, which was also previously observed by others [24], [28].

V. Related Work

A substantial effort in our community has been devoted to optimize shortest path computation [5], [7], [10]. We argue that the competition among various shortest path algorithms is no longer how computationally efficient they are. Instead, it is more about how accurate edge weights they have. A shortest path algorithm with an *acceptable* processing time and accurate edge weights would be favored over another algorithm that has faster processing time but uses inaccurate edge weights.

Indeed, the wide spread of GPS-enabled devices and subsequently the availability of trajectory data they generate has made it possible for the community to approach the problem of edge weight estimation, also known as link travel time (LTT) problem.

An influential work in this space is the one by Idé and Kato [12] in which they aimed at predicting travel time for any pair of origin destination on a map. However, they focus on inferring overall travel time of a route, as opposed to accurately inferring individual edge weights. Two pieces of work close to `MapReuse` are [24], [31]. Both of those works, rely under the assumption that weights (speeds) of spatially close road segments are similar, and hence add to the regression cost a regularizer which enforces similar weights to spatially close edges. While such approach delivers reasonably accurate results in simulated networks, it is unclear how it would perform in real life settings where spatially close roads do not necessarily have similar speeds.

The problem of static weights has led to a series of papers that allow the expression of time-dependent attributes in transportation systems in general [3], and road networks in particular [17], [28] For instance, Zheng and Ni proposed a time-dependent trajectory regression on road networks [31].

Several papers use trajectory data to find fastest routes in a road network which often requires the computation of edge travel times as an intermediate step [1], [6], [13], [27], [30]. Yuan et al. tackle the problem of finding the fastest route in a city for any triplet of source, destination, and departure time [30]. Authors use dense trajectory data generated by over 33K taxis to create a dynamic road network in which the time needed to traverse each edge is time-dependent and location-variant, i.e., the temporal dynamics of edges depend on their location. Unlike [21], [31] where edge travel times are considered as constants function of time of the day, the approach in [30] deceives different time intervals for different edges, and within each interval, the travel time is modeled as a distribution rather than a single value. Inference of future travel times on similar dynamic networks is enabled using Markov chains in [29].

Yang et al. propose to use the "incomplete" trip information for complete weight annotation of road networks [28]. The objective here is not to accurately capture trip travel times, but to infer travel costs of all edges of the road network when only few edges are covered by the trajectory data. This is a serious problem that many approaches tackle by simply focusing on "hot" (a.k.a landmarks) segments where substantial data is available [16], [30], and setting the weights of the remaining edges to zero, which is incoherent from a transportation point of view. The solution proposed in [28] consist in modeling the problem as a regression problem with an objection function that incorporates PageRank based spatial regularizers, which is quite in line with earlier works developed in [12], [31]

Many other papers tackled related problems. For instance, [2], [6], [18] are representative of probabilistic (uncertain) edge weights inference. Works in [9], [14], [26] aim at estimating travel time and/or speeds from sparse trajectory data. Instead of inferring edge weights, authors of [8], [27] propose new frameworks to deal directly with paths, avoiding splitting trajectories into small fragments.

VI. Conclusion

In this paper we showed that coarse path-level information available in most commercial maps engines can be dissected into fine-grained information which captures traffic-aware travel times on every road segment of the city. We built the system `MapReuse` which can infer travel times on each road segment in a large city using several thousand calls to commercial maps directions API. Our empiric analysis in four large metropolitan areas shows that reusing travel time info derived by `MapReuse` has very high accuracy and that `MapReuse` is able to reproduce path-travel times reported by the commercial map within a very small margin of error. We believe that the traffic layer derived by `MapReuse` can be reused by a countless array of applications which require traffic awareness.

We would like to conclude by noting that there are several questions which remain open.

Open question 1: Country-level `MapReuse`. Current version of `MapReuse` does traffic inference on the city-sized road networks. Extending it to larger regions, say country or a continent, is left for future work.

Open question 2: OD-pair selection. In the current version of `MapReuse` the selection of origin-destination pairs is fairly random, which results in many redundant paths being queried unnecessarily. We strongly believe that a more focused OD-pair selection could substantially reduce the number of API calls while maintaining the high accuracy. Minimizing the number of API calls is crucial from the end-user point of view, as it is the single parameter which controls the cost of data acquisition.

Open question 3: knowledge transfer over consecutive time-slots. Currently `MapReuse` trains the model using the API calls which coincide with a given timestamp. We believe that using information from the API calls in a consecutive times-slots may substantially reduce number of API calls needed to capture traffic information dynamics both in spatial and temporal domains. For example, over night traffic conditions rarely change, and it may not be necessary to probe independently every hour between $10pm$ and $6am$. Understanding how to transfer knowledge over time could substantially reduce the number of API calls and remains an open question.

References

[1] S. Aljubayrin and et al., "Finding non-dominated paths in uncertain road networks," in *Proceedings of ACM SIGSPATIAL 2016*.

[2] M. Asghari and et al., "Probabilistic estimation of link travel times in dynamic road networks," in *Proceedings of ACM SIGSPATIAL 2015*.

[3] P. Bakalov, E. Hoel, and W.-L. Heng, "Time dependent transportation network models," in *In Proceedings of IEEE ICDE 2015*.

[4] U. T. BUREAU, "Topologically integrated geographic encoding and referencing system," 2005.

[5] L. Chang, J. X. Yu, L. Qin, H. Cheng, and M. Qiao, "The exact distance to destination in undirected world," *The VLDB Journal*, 2012.

[6] U. Demiryurek and et al., "Online computation of fastest path in time-dependent spatial networks," in *Proceedings of SSTD 2011*.

[7] J. Dibbelt and et al., "Fast exact shortest path and distance queries on road networks with parametrized costs," in *Proceedings ACM SIGSPATIAL 2015*.

[8] J. D. et al., "Path cost distribution estimation using trajectory data," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 85–96, 2016.

[9] Z. L. et al., "Mining road network correlation for traffic estimation via compressive sensing," in *IEEE Transactions on Intelligent Transportation Systems, 2016*.

[10] R. Geisberger and et al., "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *International Workshop on Experimental and Efficient Algorithm, 2008*.

[11] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *Ieee Pervas Comput*, vol. 7, no. 4, pp. 12–18, 2008.

[12] T. Idé and S. Kato, "Travel-time prediction using gaussian process regression: A trajectory-based approach," in *Proceedings of SIAM SDM 2009*.

[13] L. Li and et al., "Go slow to go fast: minimal on-road time route scheduling with parking facilities using historical trajectory," *The VLDB Journal*, 2018.

[14] Y. Li and et al., "Urban travel time prediction using a small number of gps floating cars," in *Proceedings of ACM SIGSPATIAL 2017*.

[15] D. Luxen and C. Vetter, "Real-time routing with openstreetmap data," in *Proceedings ACM SIGSPATIAL 2011*.

[16] S. Mridha, N. Ganguly, and S. Bhattacharya, "Link travel time prediction from large scale endpoint data," in *Proceedings ACM SIGSPATIAL 2017*.

[17] T. Nakata and J. Takeuchi, "Mining traffic data from probe-car system for travel time prediction," in *Proceedings of ACM SIGKDD 2004*.

[18] B. Pan, U. Demiryurek, and C. Shahabi, "Utilizing real-world transportation data for accurate traffic prediction," in *In Proceedings of IEEE ICDM 2012*.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[20] R. Pendyala and et al., "Integrated land use-transport model system with dynamic time-dependent activity-travel microsimulation," *Transportation Research Record: Journal of the Transportation Research Board*, no. 2303, pp. 19–27, 2012.

[21] D. Pfoser and et al., "Dynamic travel time provision for road networks," in *Proceedings of ACM SIGSPATIAL 2008*.

[22] R. Stanojevic, S. Abbar, and M. Mokbel, "W-edge: Weighing the edges of the road network," Technical Report, https://ds.qcri.org/people/rstanojevic/wedge_TechReport.pdf, 2018.

[23] ——, "W-edge: Weighing the edges of the road network," in *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2018, pp. 424–427.

[24] T. I. M. Sugiyama, "Trajectory regression on road networks." in *AAAI 2011*.

[25] J. van den Berg, B. Köbben, S. van der Drift, and L. Wismans, "Towards a dynamic isochrone map: Adding spatiotemporal traffic and population data," in *LBS 2018: 14th International Conference on Location Based Services*. Springer, 2018, pp. 195–209.

[26] Y. Wang, Y. Zheng, and Y. Xue, "Travel time estimation of a path using sparse trajectories," in *Proceedings of ACM SIGKDD 2014*.

[27] B. Yang and et al., "Pace: a path-centric paradigm for stochastic path finding," *The VLDB Journal*, 2018.

[28] B. Yang, M. Kaul, and C. S. Jensen, "Using incomplete information for complete weight annotation of road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 5, pp. 1267–1279, 2014.

[29] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proceedings ACM SIGKDD 2011*.

[30] ——, "T-drive: Enhancing driving directions with taxi drivers' intelligence," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 220–232, 2013.

[31] J. Zheng and L. M. Ni, "Time-dependent trajectory regression on road networks via multi-task learning." in *AAAI 2013*.

[32] F. Z. H. V. Zuylen, "Urban link travel time estimation based on sparse probe vehicle data," in *Transportation Research Part C: Emerging Technologies 2013*.