

# STAD: Spatio-Temporal Adjustment of Traffic-Oblivious Travel-Time Estimation

Sofiane Abbar

*Qatar Computation Research Institute  
HBKU*

Doha, Qatar  
sabbar@hbku.edu.qa

Rade Stanojevic

*Qatar Computation Research Institute  
HBKU*

Doha, Qatar  
rstanojevic@hbku.edu.qa

Mohamed Mokbel

*Qatar Computing Research Institute  
HBKU*

Doha, Qatar  
mmokbel@hbku.edu.qa

**Abstract**—Travel time estimation is an important component in modern transportation applications. The state of the art techniques for travel time estimation use GPS traces to learn the weights of a road network, often modeled as a directed graph, then apply Dijkstra-like algorithms to find shortest paths. Travel time is then computed as the sum of edge weights on the returned path. In order to enable time-dependency, existing systems compute multiple weighted graphs corresponding to different time windows. These graphs are often optimized offline before they are deployed into production routing engines, causing a serious engineering overhead. In this paper, we present STAD, a system that adjusts – on the fly – travel time estimates for any trip request expressed in the form of origin, destination, and departure time. STAD uses machine learning and sparse trips data to learn the imperfections of any basic routing engine, before it turns it into a full-fledged time-dependent system capable of adjusting travel times to real traffic conditions in a city. STAD leverages the spatio-temporal properties of traffic by combining spatial features such as departing and destination geographic zones with temporal features such as departing time and day to significantly improve the travel time estimates of the basic routing engine. Experiments on real trip datasets from Doha, New York City, and Porto show a reduction in median absolute errors of 14% in the first two cities and 29% in the latter. We also show that STAD performs better than different commercial and research baselines in all three cities.

**Keywords**—Travel time estimation, traffic analysis, routing engines, transportation planning, trip duration.

## I. INTRODUCTION

Real-time estimation of travel time is in the heart of modern transportation systems, spanning applications such as ride-sharing [1], driver dispatching [2], [3], and fleet management [4]. For example, taxi and ride-sharing businesses heavily rely on travel time estimates for several core functionalities including route optimization, fare estimation, surge calculation, and taxi dispatching. With the abundance of floating vehicle data in the form of trips and GPS trajectories, it became possible to accurately estimate the travel time of trips. Three main schemes have been devised: segment-based, path-based, and origin-destination based. In segment-based approaches, GPS data is used to compute travel times for individual road segments. The travel time of a path is nothing but the sum of weights of its constituent edges [5]. Path-based approaches aim at estimating travel times for sub-paths instead of individual

segments, which allows them to capture some important inter-link transitions such as delays at junctions [6]. Finally, origin-destination (OD) based approaches aim at estimating travel times without computing paths at all [7]. Despite this variety, traditional segment-based techniques are still preferred in large production and commercial systems, including Google Maps, HERE Maps, Apple Maps, TomTom, and MapBox [8]. The reason is that most shortest-paths (routing) algorithms are optimized to work on directed graphs with static edge weights (travel times).

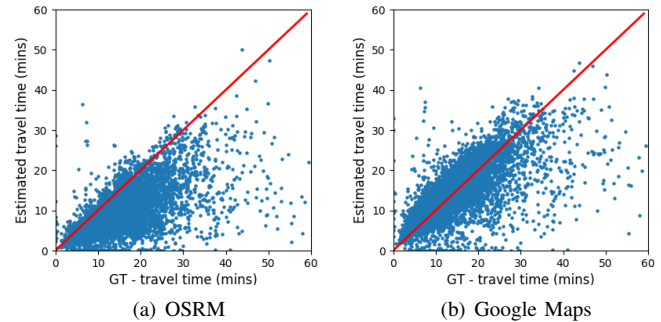


Fig. 1. Scatter plots of actual trip travel time (x-axis) and estimated travel time (y-axis) by OSRM and Google Maps on the same set of 7750 taxi trips in Doha.

All routing engines that implement these techniques are imperfect to different degrees. Figure 1 illustrates these imperfections by plotting travel time estimates produced by two routing engines, namely OSRM [9] (Fig 1(a)) and Google Maps (Fig 1(a)), compared to the actual trip duration reported for 7,750 taxi trips in Doha. Estimates were requested just before the trips started. Understandably, OSRM tends to underestimate travel times because it lacks traffic data. One way to fix this issue is to integrate time-dependency into OSRM, which requires the availability of large traffic data as well as a good knowledge on how to learn edge weights for different time windows, before one could deploy several instances of the routing engine that correspond to different time windows of interest (e.g., 24 hours of the day) [5]. However, even the most sophisticated time-dependent routing engine can engender non negligible errors as shown in Figure 1(b), in which we clearly

see that even the premium Google Maps service that accounts for both historical and live traffic yields significant errors when it comes to estimating travel times.

In this paper we describe a system, named STAD, that combines the best of segment-based and OD-based techniques to produce more accurate travel time estimates. STAD aims to quantify the spatio-temporal imperfections of any routing engine before it adjusts them to produce substantially more accurate travel time estimates. Besides, STAD can turn any static routing engine not designed for time-dependency into a full-fledged time-dependent engine with minimal sparse trips data and engineering overhead. Finally, STAD makes it easy for software developers, transport engineers, and researchers to develop in-house routing engines that can respond to live traffic data, whenever it is available to them.

The main idea of STAD is to leverage well-known spatial and temporal characteristics of road traffic modeled as features in the machine learning module to learn personalized adjustments of travel times for any trip request  $q_i$  in the format  $\langle o_i, d_i, t_i \rangle$  where  $o_i$  and  $d_i$  are respectively the origin and destination locations, and  $t_i$  is the desired departure time. STAD consists of two components. The first one, is a routing engine, possibly traffic oblivious, which is optimized to generate free-flow travel time estimates (e.g., OSRM). The second component is a machine learning (ML) based module that mines sparse trip data to find out how to effectively adjust the travel time estimates of the base routing engine in a way that realistically reflects traffic dynamics related to the time of departure  $t_i$  as well as  $o_i$  and  $d_i$  locations. Intuitively, what STAD aims to do is the following:

**Example:** Assume that a user – *Leila* – wants to go from home to work, the basic, free-flow, routing engine would return a decent path (route) with a travel time  $\tau^{FF} = 16mins$  and a total driving distance  $l = 12km$ . However, looking the trip query, STAD would infer that *Leila* is departing at 07:15am, that she lives in *West Bay* which is in the city center, and works at *Education City* which is outside of city center. Historical trips data, would allow STAD to infer that *West Bay* is fairly congested during morning rush hours on weekdays, which allows it to adjust  $\tau^{FF}$  with a  $6mins$  offset, setting the travel time estimate of *Leila*’s trip request to  $22mins$  instead. This is made possible by designing a set of features modeling the spatial and temporal traffic dependencies.

We also present STAD\*, a variant of STAD that is capable of integrating (near) live traffic data into the process of adjusting travel time estimates. Live traffic allows STAD\* to pick up unexpected traffic events as accidents, road works, or any unusual congestion not seen in the historic trip data used to train the ML module. This is made possible by tracking a global and per zone congestion indices ( $CI$ ) computed as the ratio between the speeds achieved by recent trips (e.g., last 10 minutes) versus expected speeds learnt from historic trips data for the same time window.

**Example (Cont’d):** Going back to *Leila*’s example, assuming that historic trip data reveals an expected (mean) speed of  $45km/h$  for the time window 07:00-08:00am. However, the

recently completed trips – in the previous  $10mins$  prior to *Leila*’s query (i.e., 07:05am - 07:15am) – reported an average speed of  $60km/h$  due school vacation. Then, STAD\* can take this information into account and produce a lower estimated travel time of  $19mins$  instead of  $22mins$ .

The experimental evaluations we conducted using real taxi trip datasets from Doha (Qatar), NYC (US), and Porto (Portugal) demonstrate the effectiveness of STAD when compared to different baselines from industry (Google Maps) and research (KNN [10], [11]). Indeed, our system achieves a median relative percentage error of 16.5% compared to 18.55% for KNN and 18.40% for Google Maps in Doha. In terms of median absolute errors, STAD’s travel time estimation are  $\pm 126$  seconds of ground truth travel times in Doha, which is 13% more accurate than KNN ( $\pm 144$  seconds) and 14% better than Google Maps ( $\pm 146$  seconds). Similarly in other cities, STAD’s travel time estimates are 15% more accurate than KNN in NYC and 29% in Porto. Experiments also show that live traffic data can improve the accuracy of travel times as the median absolute error of STAD\* is 5 seconds lower than that of STAD. For transparency and reproducibility of results, we voluntarily share on Github<sup>1</sup> the source code of different algorithms and baselines developed along with datasets used.

*Roadmap.* The remainder of the paper is organized as follows. Section II depicts the general architecture of STAD. Section III defines our general concepts and provides a formal definition of the travel time estimate adjustment problem. Section IV introduces – STAD – a system capable of adjusting free-flow travel-time estimates. In Section V we present STAD\*, a variant of our system capable of integrating (near) live traffic data. Section VI presents the results of our experimental evaluation on three real traffic datasets. Section VII described the related work. Section VIII concludes the paper with some remarks and future directions.

## II. ARCHITECTURE

The general architecture of STAD is depicted in Figure 2 which shows the two main components of the system: A base routing engine and a machine learning module for spatio-temporal adjustment of travel time estimates.

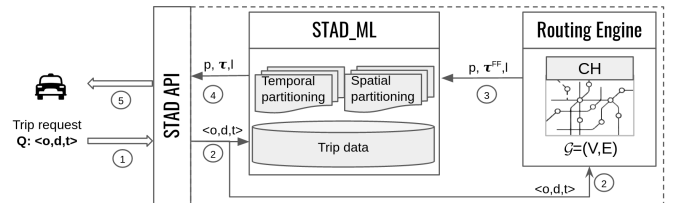


Fig. 2. High-level overview of STAD architecture.

The workflow of STAD is as follows. Upon the reception of a trip request  $q = (o, d, t)$  where  $o$  is the origin location,  $d$  is the destination, and  $t$  is the departure time, STAD API forwards the request to both components. At this stage, the

<sup>1</sup><https://github.com/vipyong/stad>

machine learning module will simply extract some spatial and temporal features from the request itself. This includes for instance the geographic zones to which  $o$  and  $d$  belong, as well as the hour of the day and day of the week by decoding  $t$ . At the same time, the routing engine generates a triplet  $(p, \tau^{FF}, l)$  where  $p$  is a path between  $o$  and  $d$ ,  $\tau^{FF}$  is the estimated travel time (free-flow), and  $l$  is the path length (driving distance). The triplet is pushed to the machine learning module, which combines  $\tau^{FF}$  and  $l$  with the previously extracted spatial and temporal features to compute an accurate travel time  $\tau$ , which is an adjustment of  $\tau^{FF}$ , that is returned to the client along with other route related information.

This architecture enables STAD to simply yet comprehensively support time-dependent travel time estimation and routing queries. It also makes it easy to digest live traffic data into these processes by compounding all these aspects into the machine learning module.

### III. BACKGROUND AND PROBLEM FORMULATION

We define here after some basic concepts related to our proposal before we formalize our problem.

**Definition 1. Road Network.** A road network is represented as a directed weighted graph  $\mathcal{G} = (V, E)$ .  $V$  is a set of nodes and  $E$  is the set of directed weighted edges (road segments). Each edge  $e_i$  comes with a length and possibly a set of intermediary points representing its geometry, a max speed value, and a type of road category. In this work, only length is considered. Each node  $v_i \in V$  is associated with a pair of (latitude, longitude) coordinates to accurately position it on the map.

**Definition 2. Path.** A path is a sequence of connected consecutive edges  $p = \{e_1, e_2, \dots, e_k\} | e_i \in E$ . The length of a path  $p_i$  is the sum of lengths of its edges, and is denoted  $l_i$ .

**Definition 3. Trajectory.** A trajectory  $tr_i = \{s_1^i, s_2^i, \dots, s_n^i\}$  is a sequence of timestamped positional sample points ( $s_j^i$ ) generated by GPS enabled devices of floating vehicles. Each point  $s_j$  comes with spatial coordinates ( $lon_j, lat_j$ ) and a timestamp  $t_j$ . Trajectories can be map-matched to  $\mathcal{G}$  to find their corresponding paths.

**Definition 4. Trip.** A trip ( $\Gamma$ ) is a tuple of the following attributes:  $\Gamma_i = (o_i, d_i, t_i, \tau_i, tr^i, p_i)$ .  $o_i$  and  $d_i$  denote the origin and destination locations respectively;  $t_i$  and  $\tau_i$  denote the departure time and the travel time respectively;  $tr^i$  and  $p_i$  denote the trajectory and its corresponding path respectively if available.  $\mathcal{T} = \{\Gamma_1, \dots, \Gamma_m\}$  is the set of trips available to us.

**Definition 5. Routing Engine** A routing engine is an optimized system built on top of the road graph  $\mathcal{G}$ , capable of executing basic routing operations such as: navigation (for shortest paths), map-matching (for most likely path covering a trajectory), distance matrix (for tables of travel times and/or distances between sets of departures and destinations). We use hereafter OSRM [9], a highly regarded open source project that many commercial companies use for routing operations.

**Problem Definition.** At a high level, our problem can be announced as follows: Given a road network graph  $\mathcal{G}$ , a set of trips data  $\mathcal{T}$ , and a trip request  $q = \langle o, d, t^d \rangle$  where  $o = (lon^o, lat^o)$  and  $d = (lon^d, lat^d)$  are origin and destination locations, and  $t^d$  is the departure time, find the most likely travel time  $\hat{\tau}$  for  $q$ . However, given the two stages nature of our system, we can reformulate the problem as follows: Given a basic routing engine, a set of trips  $\mathcal{T}$ , and a trip query  $q$ , first compute the free-flow travel time  $\tau^{FF}$  and path length  $l$  for  $q$  using the basic routing engine; Next, learn to adjust  $\tau^{FF}$  to reflect actual traffic patterns covered by  $\mathcal{T}$  and produce more accurate travel time estimate  $\hat{\tau}$ .

## IV. STAD: SPATIO-TEMPORAL ADJUSTMENT OF TRAVEL TIME ESTIMATES

In this section, we describe our system, STAD, for Spatio-Temporal ADjustment of traffic oblivious travel times. We first give a brief overview of the system. Then, we explain how space and time are partitioned to capture varying spatio-temporal impact of traffic on travel time estimation. Next, we discuss the features we used and the choice of the machine learning algorithm. Finally, we describe the online adjustment process of travel times.

### A. General overview

STAD consists of two main components: (i.) A base routing engine used to produce free-flow paths and travel time estimates between pairs of locations. One can think of this as simply running Dijkstra algorithm for shortest path on a directed weighted graph representing the road network, where weights are traversal times of edges derived from length and default speeds. (ii.) A machine learning module that uses spatial and temporal features to adjust free-flow travel time estimates to traffic patterns captured in the trips data. These components are first used offline to learn the best parameters possible for the adjustment of travel times in a given city, then online to adjust the travel time of any trip request.

### B. Spatial and temporal partitioning

Intuitively, in the free-flow scenario where there is no traffic at all, the travel time of a trip is governed by two parameters: the length of the trip and the maximum speed allowed on different road segments. With traffic, there are two more parameters that impact the travel time: origin and destination locations, and departure time. In other words, the overhead caused by traffic on the free flow travel time varies significantly from an area to another, from an hour to another, from a day to another. Thus, the need to split space and time into smaller units that allow capturing the varying impact of traffic (overhead) in time and space. There are different way the space, i.e., a city in our case, can be partitioned. One common practice consists in dividing the space into a grid of equal-sized squares, e.g.,  $100m \times 100m$ . This technique has two major drawbacks: First, it destruct the existing road network boundaries by crossing roads and junctions at random locations. Second, it does not account for spatial population

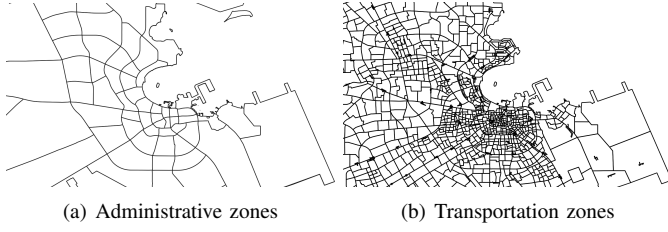


Fig. 3. Example of administrative vs. transportation zones in Doha

and road distributions, yielding zones with no roads or no trips. Hence, we opt for the use of administrative boundaries to split the space into zones. This practice is very common in transportation research, and such data is abundantly available online<sup>2</sup>. Administrative zones are usually generated in a way that preserves the natural road network boundaries and population distribution. It is worth mentioning that there are more specialized partitioning schemes such as transportation area zones (TAZ), which are more adequate for transportation related studies. However, given that TAZ are not always publicly available, we use administrative boundaries in this paper and evaluate the impact of different partitioning schemes in Section VI-G. Figure 3 shows samples of zones in Doha with different granularities. Once the space is partitioned, it is easy to assign trip's origin and destination locations to specific zones.

Similarly to space, time is also often partitioned into small intervals, called time windows. The size of the intervals is typically between 15 mins to 1 hour, depending on the amount and coverage of the trip data available. The finer the granularity, the higher the sparsity. In hour case, we assign each trip  $\Gamma_i$  to the time window corresponding to its starting time. We also derive a set of temporal features as shown in the next section.

### C. Feature engineering

Now, we enumerate the different features that STAD uses to learn good spatio-temporal adjustments of travel times. Some of these features are spatial, others are temporal, and some of them are related to historical traffic conditions that capture recurrent dynamics such as morning and evening commutes in different zones.

- $z_o$  – *origin zone* : this is a unique "alphanumeric" identifier of the zone into which falls the origin location of the trip. Inferred from  $o = (lon^o, lat^o)$ .
- $z_d$  – *destination zone* : identifier of the destination zone. Inferred from  $d = (lon^d, lat^d)$ .
- $h_d$  – *hour of the day*: this is an integer in  $\{0, 2, \dots, 23\}$  representing the hour of the day in local time; 00 is for midnight.
- $d_w$  – *day of the week*: an integer in  $\{1, 2, \dots, 7\}$ ; 1 is for Monday and 7 is for Sunday.
- $h_w$  – *hour of the week*: an integer in  $\{0, 2, \dots, 167\}$  that captures the hours of the week (24x7).

<sup>2</sup><https://www.geofabrik.de/data/>

- $\tau^{ff}$  – *free-flow travel time*: the travel time returned by RE for a given pair  $o, d$ . Expressed in seconds.
- $l_d$  – *driving distance*: the length of the path returned by RE for the pair  $o, d$ . Expressed in meters.
- $l_h$  – *Haversine distance*: this is the straight line distance between  $(o, d)$ . Expressed in meters.
- $p_c$  – *path complexity*: this is the ratio between the  $l_d$  and  $l_h$ ;  $p_c = l_h/l_d$ . The closest the value to one, the simplest is the driving route between  $o$  and  $d$ .
- $tr_{z_o}^*$  – *trips departing from  $z_o$* : ratio of trips departing from  $z_o$  to all trips. This captures how busy are the outgoing roads from the origin zone,  $z_o$ .
- $tr_{z_d}^*$  – *trips arriving at  $z_d$* : ratio of trips arriving to  $z_d$  to all trips. Captures the business of incoming roads to the destination zone,  $z_d$ .

### D. Model Selection

For each trip  $\Gamma_i = (o_i, d_i, t_i, \tau_i, tr^i, p_i) \in \mathcal{T}$ , we build a vector feature  $x_i = \langle z_o, z_d, h_d, d_w, h_w, \tau^{ff}, l \dots, tr_{z_o}^*, tr_{z_d}^* \rangle$  using all features defined in the previous section. Next, we create for each  $\Gamma_i$  a pair  $(x_i, \tau_i)$  where  $x_i$  is the feature vector and  $\tau_i$  is the actual travel time reported for  $\Gamma_i$ . The objective now is to find a function  $F$  that maps  $x_i$  to  $\tau_i$ , such that the following quantity is minimised:

$$\frac{1}{|\mathcal{T}|} \times \sum_{tr_i \in \mathcal{T}} L(\tau_i, F(x_i)),$$

where  $L$  is a loss function that captures the errors between exact travel times and adjusted ones. This is a classical supervised machine learning framework, regression to be more specific, where one can try different algorithms and pick the best achieving one. However, given that our features are a mix of numerical and categorical, we opted for the tree-based ensemble method called Gradient Boosting [12]. We know that categorical features can be converted into binary numerical features that work with more algorithms such as SVM and linear regression, however, when the number of categories is high (which is the case of number of zones for instance), this process results in very sparse feature vectors with high dimensionality which requires non trivial treatment via regularizers. We set the loss function  $L$  to be least squares and fine-tune the Gradient Boosting Regressor to find the best combination of hyper parameters, i.e., number of trees and max depth of each tree. At the end of this step, we obtain a model  $\mathcal{M}$  that is capable of predicting accurate traffic-aware travel time estimates  $\hat{\tau}$  for any feature vector  $x_i$ .

### E. Online querying

The online step of STAD is quite simple and efficient (See algorithm 1). This is important not to cause any overhead to the already optimized routing engine which can be deployed in production, i.e., serves thousands of routing requests per second. Given a trip request  $q = \langle o, d, t^d \rangle$  with origin, destination, and departure time, and a prediction model  $\mathcal{M}$ , we first create the feature vector by mapping locations to zones (lines 2–3), extracting temporal features from departure time

(line 4), and querying the basic routing engine for free-flow travel-time estimates and driving distance. Next, we compound all these features into one vector and run the model method (line 6.) Other features defined in the previous sections can easily be added here.

---

**Algorithm 1** Online phase of STAD

---

```

1: Input:  $\mathcal{M}$  - ML predictive model trained offline.  $q = \langle o, d, t^d \rangle$  - a trip request
2:  $z_o = locate(o)$ ; {find area zone of origin}
3:  $s_d = locate(d)$ ; {find area zone of destination}
4:  $h_d, h_w, d_w = extract\_temporal\_features(t^d)$ 
5:  $\tau^{FF}, l = RE(o, d)$ ; {generate basic free-flow information for  $q$ }
6:  $\hat{\tau} = \mathcal{M}.predict(z_o, z_d, h_d, h_w, d_w, \tau^{FF}, l)$ 
7: return  $\hat{\tau}$ 

```

---

## V. STAD\*: LIVE TRAFFIC INTEGRATION

While STAD is good at capturing recurrent traffic patterns such as morning and evening commutes, and weekends vs. weekdays, it is not well suited to deal with unexpected events such as sudden congestion, accidents, processions, or severe weather conditions such as heavy rain. Thus, in the case live traffic is available us, we devise STAD\*, a variant of STAD capable of integrating (near) live traffic conditions into the adjustment process of travel time estimations.

One simple way to take into account live traffic information is to compare the current traffic status to the expected one from historical data. This can be achieved, by monitoring the average speed of vehicles. We propose to monitor the traffic level via a congestion index  $\mathcal{CI}$ . Assuming that time is partitioned into windows, i.e.,  $\mathcal{TW} = \{tw_0 \dots tw_{24}\}$  to capture hourly traffic patterns for a typical day, we could easily compute offline the average speed observed at each time window. At query time, assuming that we would like our system to update to live traffic every 10mins, we compute the average speed of all trips ( $\mathcal{T}_c$ ) completed in the last 10mins and compare it to the expected speed from the time window that corresponds to the last 10 minutes. Equation (1) gives the formula used to compute  $\mathcal{CI}$ . A value greater than 1.0 indicates a traffic that is less congested than expected whereas a value lower than 1.0 indicates a traffic level that is more congested than expected.

$$\mathcal{CI}_{tw} = \frac{\frac{1}{|\mathcal{T}_c|} \sum_{\Gamma_i \in \mathcal{T}_c} l_i / \tau_i}{\frac{1}{|\mathcal{T}_{tw}|} \sum_{\Gamma_j \in \mathcal{T}_{tw}} l_j / \tau_j} \quad (1)$$

$\mathcal{CI}$  makes it easy for instance to distinguish an 8am of a busy Monday from the fluid 8am of a quite Sunday. Even in cases where time is partitioned into finer windows (e.g.,  $24 \times 7 = 168$ ) to capture weekly patterns,  $\mathcal{CI}$  allows to distinguish a 10am of a busy Tuesday from a 10am of calm Tuesday that coincides with a school break for instance. Depending on the values of  $\mathcal{CI}$ , STAD\* will learn to increase or decrease the adjusted travel time  $\hat{\tau}$  to reflect the current traffic condition. The following additional features can be derived from Equation (1):

- $\mathcal{CI}_{h_d}$  – *global congestion index - day*: Congestion index by comparing current speeds to average speeds of the hour of the day ( $h_d$ ) to which the departure time of the request falls into.
- $\mathcal{CI}_{h_w}$  – *global congestion index - week*: Congestion index by comparing current speeds to average speeds of the hour of the week ( $h_w$ ) to which the departure time of the request falls into.

These two features are added to the ones defined in the previous section in order to train a new predictive model  $\mathcal{M}^*$  for STAD\*.

## VI. EXPERIMENTS

In this section, we describe our evaluation setup, algorithms and baselines, metrics, data we used for different experimental evaluations.

### A. Algorithms and Baselines

In order to assess the importance of different features on the accuracy achieved by our system, STAD, we evaluated four different variants: STAD<sub>t</sub> which only uses temporal features (i.e.,  $\tau^{ff}$ ,  $l_d$ ,  $h_d$ ,  $d_w$ , and  $h_w$ ), STAD<sub>st</sub> which uses spatio-temporal features (i.e.,  $\tau^{ff}$ ,  $l_d$ ,  $h_d$ ,  $d_w$ ,  $h_w$ ,  $z_o$ , and  $z_d$ ), STAD<sub>all</sub> which uses all features described in Section IV, and finally STAD\* that integrates live traffic data. Without loss of generality, STAD and STAD<sub>st</sub> are used interchangeably in the remaining of this section. We compare our systems to three different baselines:

1) *RE*: We use OSRM [9] as our default traffic oblivious routing engine (RE). OSRM uses the road network extracted from OpenStreetMaps<sup>3</sup> to build its internal graph. The system relies on some prior knowledge of traffic in order to compute free-flow weights to each road segment. These weights are generated based on metadata associated with road segments, including type of road (e.g., highway), posted speed (e.g., 50kmph), and penalty scores for different types of turns (e.g., left-turn, U-turn).

2) *KNN*: In addition to RE, we implemented KNN, a nearest neighbors based algorithm developed by Wang et al. [10], [11]. Similar to STAD, KNN combines spatio-temporal aspects of trips to predict travel time of any trip query  $q = (o_q, d_q, t_q)$ . The algorithm assumes a collection of historical trips  $\mathcal{T}$  each of which is defined as 5-tuple  $\Gamma_i = \langle o_i, d_i, t_i, l_i, \tau_i \rangle$  where  $o_i, d_i$  are origin and destination locations,  $\tau_i, l_i$  are respectively the travel time and the distance of the trip, and  $t_i$  is the departure time. Given a query  $q$ , KNN computes the expected travel time of  $q$  as follows:

$$\hat{q} = \frac{1}{|\mathcal{N}(q)|} \sum_{\Gamma_i \in \mathcal{N}(q)} \tau_i \times \frac{V(t_i)}{V(t_q)},$$

where  $\mathcal{N}(q)$  is the set of nearest neighbor trips to  $q$ . A trip  $\Gamma_i$  is considered as a nearest neighbor to  $q$  if their origins and destinations are close enough. In our adaptation, we assume two trips as nearest neighbors if their origins and destinations

<sup>3</sup><https://www.openstreetmap.org>

fall into the same spatial zones respectively. KNN splits time into 168 windows corresponding to the hours of a typical week, then  $V(t_i)$  (resp.  $V(t_q)$ ) is the average speed observed for all trips that started at the hour corresponding to the departure time  $t_i$  (resp.  $t_q$  departure time of  $q$ .) Speeds can easily be inferred from the distance  $l_i$  and travel time  $\tau_i$  of any trip  $\Gamma_i$

3) *Google Maps*: We also compare our findings against one of the leading commercial map engines in the market, Google Maps. Using their Direction API <sup>4</sup>, Google Maps allows developers to query for best routes for any request  $q = (o_q, d_q, t_q)$  such that  $t_q$  is sometimes in the future. Google Maps uses then historical traffic data to compute the travel times of these requests. We sampled uniformly at random 2K different trip requests that we submitted to Google Maps. Since the timestamp need to be in the future we submitted our queries in the last week of January with the timestamps falling in the second week of February. For a fair comparison, timestamps used in Google maps queries correspond to the same hour/weekday/month of each trip in our dataset. An important remark here is that even though our Doha dataset is about 2 years old, the road network as well as the traffic pattern have experienced a nontrivial change between 2018 and 2020, albeit quantifying that change is difficult and out of scope of the present paper. The data in NYC and Porto are from 2013 and we believe that traffic patterns over 6-7 years change substantially, and do not compare these 2 cities with Google Maps API in this paper.

### B. Evaluation metrics

In order to assess the accuracy and quality of travel time estimates produced by different algorithms, we use mean absolute error (MAE) and median absolute error (MedAE), which have both seconds as units. However, because the values of these metrics depend on the travel time distribution in our datasets, it important to report relative errors as well. Thus, we compute the mean absolute percentage error (MAPE) also known as mean relative error as well as the median absolute percentage error (MedAPE.) Formulas for the different metrics are defined as follows:  $MAE(Y, \hat{Y}) = 1/n \sum_{i=1, n} |y_i - \hat{y}_i|$ ,  $MedAE(Y, \hat{Y}) = median(\{|y_i - \hat{y}_i\}_{i=1, n})$ ,  $MAPE(Y, \hat{Y}) = 100 \times 1/n \sum_{i=1, n} |y_i - \hat{y}_i|/y_i$ , and finally  $MedAPE(Y, \hat{Y}) = 100 \times median(\{|y_i - \hat{y}_i|/y_i\}_{i=1, n})$

### C. Dataset description

We use trips data generated by Taxi companies operating in three different cities: Doha (Qatar), NYC (US), and Porto (Portugal). Table I presents some statistics about the datasets. The time span covered by trips varies from 1 week in 2013 for NYC, to 5 weeks in 2018 for Doha, to 1 year 2013/2014 for Porto. We also found a significant discrepancy in the distribution of travel times among the three cities. Doha for instance has a median trip duration of 13.26 minutes compared to

10.55 minutes in NYC and 7 minutes in Porto. Complementary cumulative distribution functions of travel times are illustrated in Figure 4 which shows that around 90% of trips last less than 12 minutes in Porto (rapid decay) versus 30 mins for Doha and NYC.

TABLE I  
CHARACTERISTICS OF DIFFERENT DATASETS USED IN THE EXPERIMENTAL EVALUATION.

city	dates	#trips	med. TTE	trips/day
Doha	2018/01/01-2018/02/08	748,096	13.26	19.2K
NYC	2013/11/01-2013/11/07	2,078,503	10.55	297K
Porto	2013/07/01-2014/07/01	662,614	7.0	2.5K

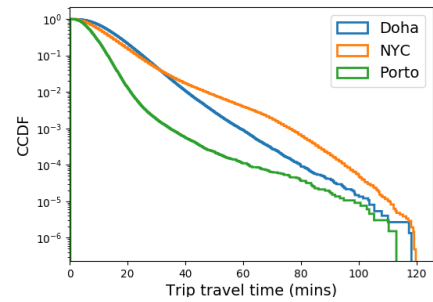


Fig. 4. Complementary cumulative distribution function of travel times estimates for Doha, NYC, and Porto (y-axis is in log scale.)

### D. Main results

We train different STAD models on 70% of the trips and test on the remaining 30%. Reported results are averages over multiple runs.

Figure 5 illustrates the main results for travel time estimation using the three cities. In the first row, we report the mean and median absolute errors achieved by different algorithms in predicting travel times. The second row reports global relative errors whereas the third one reports hourly relative errors. The foremost observation to be made here is that  $STAD_{st}$  outperforms by far all other algorithms in all three cities and in all considered metrics. Second, we clearly see that accounting for spatial features ( $STAD_{st}$ ) yields significant improvement compared to  $STAD_t$  which uses temporal features only. This is shown in Figures 5(a), 5(b), and 5(c) which report respectively a reduction in median absolute errors induced by the spatial features of 16 seconds in Doha, 6 seconds in NYC, and 7.1 seconds in Porto. The gain in relative errors (second row of Figure 5) is of two points in Doha (18.31% to 16.5%), 1.5% in Porto and 1% in NYC.

To our surprise, the simple  $STAD_t$  did better than KNN in all experimental configurations, let alone  $STAD_{st}$ . In Doha for instance, KNN achieves a median average error of 144 seconds vs. 142 seconds for  $STAD_t$  and a low 126 seconds for  $STAD_{st}$ . The absolute errors of KNN are also 15% higher than  $STAD_{st}$  in NYC and 29% higher in Porto. In terms

<sup>4</sup><https://developers.google.com/maps/documentation/directions/start>

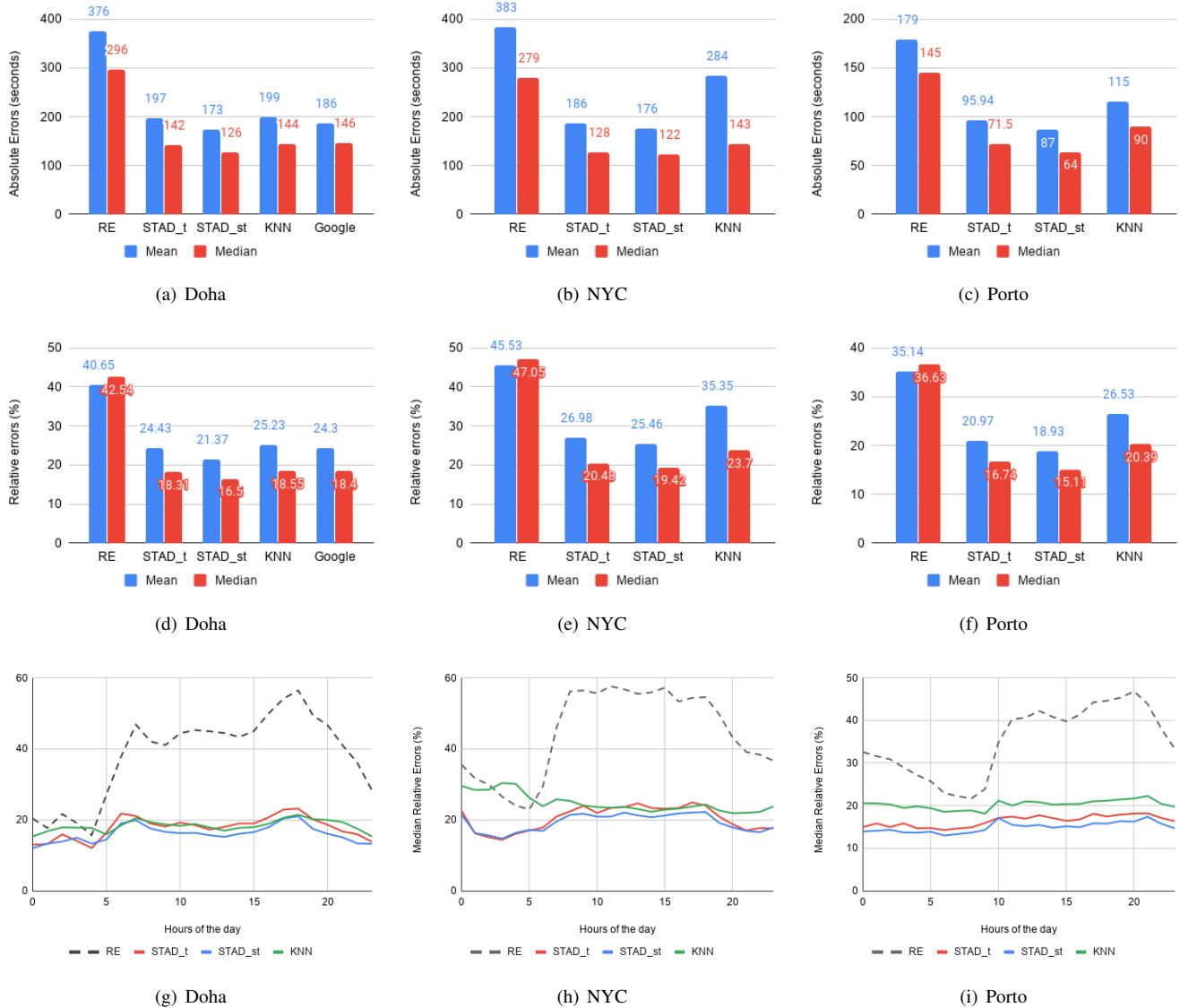


Fig. 5. Main results for travel time predictions. The first row reports mean and median absolute errors of travel time estimations for unseen trips in three different cities. The second row reports mean and median relative errors in percentages for the same trips. The third row reports hourly median relative errors.

of relative errors,  $STAD_{st}$  is 2 points lower than KNN in Doha (16.5% vs. 18.5%), 4 points lower in NYC (19.42% vs. 23.7%), and 5 points lower in Porto (15.11% vs. 20.39%). These big improvements of  $STAD_{st}$  over KNN are intriguing when we know that the latter is designed to embed the spatio-temporal aspects. Our interpretation is that relying on origin and destination to declare two trips similar is not a good idea. One could at least account for trips distance to remove outliers. Also, simply averaging travel times of similar trips is not good enough, especially when all nearest neighbor trips are considered regardless of the time at which they were made.

As expected, the experiments show that even for a state-of-the-art routing engine which incorporates a significant prior knowledge about traffic such as delays at junctions, traffic lights, turn penalties, it is very difficult to get travel time estimates right without traffic information. This is reflected in the

high median absolute percentage (relative) errors achieved by RE in Doha (42.54%), NYC (47.05%), and Porto (36.63%) as shown in Figures 5(d), 5(e), and 5(f). However, it is interesting to note that free-flow travel time estimates produced by RE are quite comparable to other methods for specific time windows (00:00 -04:00am in Doha, with a MedAPE=20%). In the case of NYC, RE outperforms KNN between 03:00-05:00am. This is worth exploring for companies who want to cut some of the cost that goes to commercial maps.

As planned, we also tried Google maps services for Doha and reported results in Figures 5(a) (absolute errors) and 5(d) (relative errors.) In terms of relative errors, Google is slightly better than KNN with a median percentage error of 18.4% compared to 18.55%. However, both version of STAD are better than Google with a clear win of  $STAD_{st}$  which achieves a median percentage error of 16.5%. Similarly, the median

TABLE II  
RESULTS OF DIFFERENT STAD VARIANTS ON DOHA DATA.

Variant	MAE(sec)	MedAE(sec)	MAPE(%)	MedAPE(%)
STAD <sub>t</sub>	196.78	141.43	24.35	18.4
STAD <sub>st</sub>	174.49	126.76	21.40	16.50
STAD <sub>all</sub>	172.14	125.42	21.01	16.33

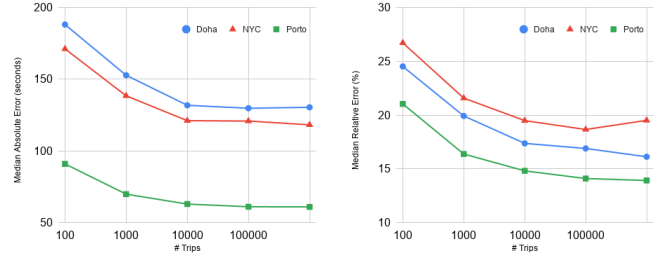
absolute error achieved by Google is 146 seconds which is quite similar to KNN with 144 seconds and STAD<sub>t</sub> with 142 seconds. But once again, STAD<sub>st</sub> shows a clear superiority with 126 seconds of median percentage error. We deliberately decided not to compare with NYC and Porto trips because they are more than 7 years old which makes it unfair comparison.

We report in the third row of Figure 5 the hourly median percentage errors of different algorithms in different cities. Each curve has 24 points corresponding to the 24 hours of the day. In the case of Doha for instance, we clearly distinguish two error spikes corresponding to morning (06:00am-07:00am) and evening (05:00pm-06:pm) commutes in which the percentage error is above 20% whereas the errors are below the bar of 20% for all other hours of the day. This behavior is partly due to the fact that traffic is more unpredictable during rush hours, when incidents tend to happen more often. In the remaining hours of a typical day, where traffic shows more stability and seasonality, all algorithms seem to do well, with a net superiority for STAD<sub>st</sub> though. The same kind of pattern is observable in NYC (Figure 5(h)) and Porto (Figure 5(i)), thought for the particular case of NYC the error pattern looks two-phased: day (07:00am to 07:00pm) where errors are above 20% and night (07:00pm to 07:00am) where errors go below 20%. This might be explained by the constant business of the city throughout the day.

Finally, we report in Table II results achieved by different version of STAD. We clearly see that accounting for spatial features yields significant improvements, i.e median absolute error of STAD<sub>st</sub>(126 secs) is 10% better compared to STAD<sub>t</sub> (141 secs). However, adding more features STAD<sub>all</sub> (125 secs) yields only 1% improvements over STAD<sub>st</sub>.

### E. Impact of number of trips

We investigate the impact of the number of trips available for training STAD on the accuracy of the predicted travel times in all cities. Figures 6(a) and 6(b) report respectively median absolute and relative errors in the three cities, function of the number of trips used in the training. The elbow shaped curves we see indicate a diminishing return property according to which the impact of adding more data reduces as the size of data increases. For instance, we observe that absolute errors drastically improve with more trips until we reach the breaking point of 10K trips. Adding more trips beyond 10K does not seem to yield any significant reduction in terms of absolute error for all three cities. However, if the objective is to optimize for median percentage errors, then it seems that 100K is the magic number of trips needed. In the case of NYC, we even see an inverse phenomenon where the percentage error



(a) Median Absolute Errors (sec.) (b) Median Relative Errors (%)

Fig. 6. The effect of number of trips available for training on the accuracy of travel time predictions of STAD<sub>st</sub> in the three cities.

increases when more trips are used, which can be due to over-fitting.

### F. Impact of real-time traffic data

When live traffic is available i.e., the possibility to access trips as soon as they are completed, it becomes possible to deploy STAD\* which incorporates congestion indices into the adjustment model. Table III shows comparative results between STAD (historical traffic model) and STAD\*. As expected, live traffic information yields significant improvements in the accuracy of travel time of about 5 seconds in median absolute error ( $\approx 4\%$  reduction). Mean absolute percentage error gains almost 1 point lowering from 21.27% for STAD to 20.38% for STAD\*.

TABLE III  
THE IMPACT OF REAL-TIME TRAFFIC DATA ON TRAVEL TIME ESTIMATION ACCURACY IN DOHA DATASET.

Variant	MAE(sec)	MedAE(sec)	MAPE(%)	MedAPE(%)
STAD	161.11	122.30	21.27	16.20
STAD*	158.31	117.37	20.38	15.75

As mentioned earlier, the real advantage of STAD\* over STAD is its ability to correct for specific events and for days that are substantially different from those seen in training on historical data. For instance, we found that STAD\* yields to significant reduction in median absolute error for trips happening on Fridays, especially for those taking place during time intervals considered as rush hours in other days (Note that Friday is a weekend in Doha and its traffic patterns are quite similar to those observed on Sunday in the west.) That is, STAD\* saves 21 seconds for trips starting at 5pm, 11 seconds for those starting at 6 pm (evening commute window). Likewise, we observe a reduction of 13 seconds for 7am and 12 seconds for 6am (morning commute.) It is worth mentioning that the evaluation setup for this comparison is different from the one we used to produce results of Table II, which explains the differences in scores obtained by STAD<sub>st</sub> in the two tables. Indeed, to account for live traffic, we needed to sort trips by departure time. We removed then the first 200K trips, used to learn historical average speeds for different time windows, which represents around 26% of all trips in Doha dataset.



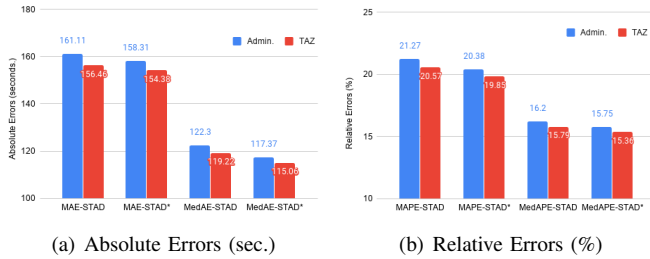


Fig. 7. Comparative results for different spatial partitioning schemes on accuracy of travel times in Doha.

### G. Impact of spatial partitioning choice

We investigate now the impact of way the space is partitioned on the accuracy of travel time estimation. We have deliberately decided to use administrative zoning as a default choice for partitioning, which can be easily obtained for almost all cities around the world. However, it is well known that transportation departments around the world have their own way of partitioning a city into geography units known as traffic/transportation analysis zones (TAZ), which takes into account population densities as well as the geometry of the road network. Thanks to our collaboration with Ministry of Transport and Communication in Qatar, we could obtain their TAZ shapefile which contains 1,839 zones. We re-assigned origin and destination locations of every trip into one of these zones, and re-run both STAD\* and STAD on the new data. We show in Figure 7 comparative results achieved using two different spatial partitioning schemes. Overall, we notice that TAZ yields better accuracy compared to regular administrative zones. The gain in median (resp. mean) absolute error is of 3 secs (resp. 5 secs) for STAD and 2 seconds (resp. 4 secs) for STAD\*. The gain in relative errors is less significant though.

## VII. RELATED WORK

The abundance of transportation data in the form of GPS trajectories and trips has opened new horizons for capturing traffic dynamics in road networks. We can distinguish three main categories of work aiming at estimating trips travel time: segment-based, path-based, and origin-destination based.

*Segment-based.* Also known as link travel time, is the most common approach in which GPS data (trips or trajectories) are used to learn the weights of individual edges of the road graph. Travel time of a path is then calculated as the sum of weights on its constituent edges [5], [13]–[18]. Several techniques have been explored to efficiently and accurately learn these weights. For instance, [5], [13], [15], [16], [19] use different types of regression (e.g., linear, ridge, Gaussian), some times with regularizers, to effectively learn the weights. Similarly, [20] uses Markov chains to infer future travel times, whereas [21] proposes to use a spatio-temporal extension of Hidden Markov models. In order to deal with the inherent data sparsity problem in which not all segments can be covered with data, several authors have adopted matrix decomposition techniques for missing values completion and travel time prediction [22]–

[24]. In most of the cases, the decomposition is done with the introduction of either some latent features for the road network (e.g., [22]) or some particular temporal regularizers (e.g., [23].)

*Path-based.* One of the main shortcomings of segment-based techniques is their failure to capture inter-link transitions such as turns at intersections or waiting at traffic lights. Some approaches tackled this problem by explicitly estimating the time spent at junctions [25], [26]. Another line of work aims at concatenating segments into (sub-)paths and estimating travel time directly for paths instead of individual segments [6], [17], [18], [27]. In [17], authors propose to use a three dimensional tensor (users, segments, time) combined with frequent trajectory patterns to find the best combination of sub-paths to use in online travel time estimation. The solution is shown to find a good trade-off between the length of sub-paths and number of trajectories traversing them. Dai et al. [27] take a different approach to the same problem and introduce the concept of hybrid graphs to accurately capture the cost distribution of paths. The idea is to learn a weighting function for paths instead of edges, then find a good approach to combine distributions of multiple sub-paths to estimate the travel time of a query path. A follow-up to this work is PACE [6] in which authors solve the problem in a more principled way by tackling two problems: first, estimation of path cost distributions using an optimal set of trajectories; second, finding the rights paths for a source-destination pair.

*OD-based.* While path-based techniques overcome the inter-link transition problem, they do introduce some difficult challenges such as finding the right sub-paths for a query trip, which is often solved in an ad-hoc mode or using heuristics. More recently, some authors looked at inferring travel time estimates for pairs of origin-destination without computing paths [7], [10], [11], [28], [29]. This is particularly relevant in online configurations where path information is not available or costly to obtain. Examples of applications where this is needed include taxi dispatching where there is a huge need to compute travel time estimates between locations of a set of vehicles and that of a customer. One of the earliest works in this space is done by Wang et al. [10], [11]. Here, authors propose to compute the travel time of a given trip defined with an origin, destination, and departure time, by looking at the travel time of historic nearest neighbor trips. Two trips are assumed to be neighbors if their origins and destinations fall respectively within a predefined radius distance. A scaling factor that captures traffic variations is then computed to allow for a weighted averaging of travel times reported by nearest neighbor trips. A more recent work by Li et al. [7] uses multi-task representation learning for arrival time estimation. The idea is to learn link representations that optimize not only for travel times but also for other related targets such as travel distance and number segments on the path. The authors propose to embed the spatio-temporal aspects of traffic via Laplacian regularization. other “unpublished” yet interesting attempts explored the use of deep neural networks such as GNNs [29] and ST-NNs [28] to travel time estimation tasks.

## VIII. CONCLUSION

We presented in this paper, STAD, a novel system capable of adjusting free-flow travel time estimates produced by any traffic oblivious routing engine to match actual traffic patterns of a city. STAD partitions the space into areas or zones, and time into windows to capture the spatio-temporal patterns of traffic. We also presented STAD\*, a version of our system that adapts the adjustment of travel time to live traffic conditions. Our experiments on real traffic datasets from three cities show that STAD yields significant improvement in terms of travel time accuracy compared to existing methods, or even to premium commercial routing engines such as Google Maps. Indeed, in the case of Doha for instance, our system achieves a median relative percentage error of 16.5%, whereas KNN based methods achieve 18.55% and Google Maps 18.40%. Also, STAD's median absolute error is 126 seconds compared to 144 second and 146 seconds for KNN and Google Maps respectively. In the future, we would like to explore the use of spatially and temporally weighted regression methods to better account the spatio-temporal non stationarity of road traffic. We are also working on releasing an end-to-end open-source system built on top of OSRM to make it easy for people and companies to get started with time-dependent routing and travel time estimation.

## ACKNOWLEDGMENT

We would like to thank the Land Transport Planning Department at the Ministry of Transport and Communication in Qatar for their meaningful engagement with our research. We would like also to thank Karwa Technologies at Taxi Mowasalat for their continuous support and fruitful collaboration.

## REFERENCES

- [1] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 2013, pp. 410–421.
- [2] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie, "T-finder: A recommender system for finding passengers and vacant taxis," *IEEE Transactions on knowledge and data engineering*, vol. 25, no. 10, pp. 2390–2403, 2012.
- [3] Z. Liao, "Real-time taxi dispatching using global positioning systems," *Communications of the ACM*, vol. 46, no. 5, pp. 81–83, 2003.
- [4] A. Cristian, L. Marshall, M. Negrea, F. Stoichescu, P. Cao, and I. Menache, "Multi-itinerary optimization as cloud service (industrial paper)," in *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2019, pp. 279–288.
- [5] R. Stanojevic, S. Abbar, and M. Mokbel, "W-edge: Weighing the edges of the road network," in *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2018, p. 424427.
- [6] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu, "Pace: a path-centric paradigm for stochastic path finding," *The VLDB Journal*, vol. 27, no. 2, pp. 153–178, 2018.
- [7] Y. Li, K. Fu, Z. Wang, C. Shahabi, J. Ye, and Y. Liu, "Multi-task representation learning for travel time estimation," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1695–1704.
- [8] D. Delling, "Route planning in transportation networks: From research to practice," in *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL 18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2. [Online]. Available: <https://doi.org/10.1145/3274895.3282802>
- [9] D. Luxen and C. Vetter, "Real-time routing with openstreetmap data," in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '11. New York, NY, USA: ACM, 2011, pp. 513–516. [Online]. Available: <http://doi.acm.org/10.1145/2093973.2094062>
- [10] H. Wang, Y.-H. Kuo, D. Kifer, and Z. Li, "A simple baseline for travel time estimation using large-scale trip data," in *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2016, p. 61.
- [11] H. Wang, X. Tang, Y.-H. Kuo, D. Kifer, and Z. Li, "A simple baseline for travel time estimation using large-scale trip data," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–22, 2019.
- [12] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.
- [13] T. Idé and S. Kato, "Travel-time prediction using gaussian process regression: A trajectory-based approach," in *Proceedings of the 2009 SIAM International Conference on Data Mining*. SIAM, 2009, pp. 1185–1196.
- [14] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "T-drive: Enhancing driving directions with taxi drivers' intelligence," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 220–232, 2011.
- [15] T. Idé and M. Sugiyama, "Trajectory regression on road networks," in *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.
- [16] J. Zheng and L. M. Ni, "Time-dependent trajectory regression on road networks via multi-task learning," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.
- [17] Y. Wang, Y. Zheng, and Y. Xue, "Travel time estimation of a path using sparse trajectories," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 25–34.
- [18] S. Aljubayrin, B. Yang, C. S. Jensen, and R. Zhang, "Finding non-dominated paths in uncertain road networks," in *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2016, pp. 1–10.
- [19] R. Stanojevic, S. Abbar, and M. Mokbel, "Mapreuse: Recycling routing api queries," in *2019 20th IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 2019, pp. 279–287.
- [20] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 316–324.
- [21] B. Yang, C. Guo, and C. S. Jensen, "Travel cost inference from sparse, spatio temporally correlated time series using markov models," *Proceedings of the VLDB Endowment*, vol. 6, no. 9, pp. 769–780, 2013.
- [22] D. Deng, C. Shahabi, U. Demiryurek, L. Zhu, R. Yu, and Y. Liu, "Latent space model for road networks to predict time-varying traffic," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1525–1534.
- [23] H.-F. Yu, N. Rao, and I. S. Dhillon, "Temporal regularized matrix factorization for high-dimensional time series prediction," in *Advances in neural information processing systems*, 2016, pp. 847–855.
- [24] A. Baggag, S. Abbar, A. Sharma, T. Zanouda, A. Al-Homaid, A. Mohan, and J. Srivasatava, "Learning spatiotemporal latent factors of traffic via regularized tensor factorization: Imputing missing values and forecasting," *IEEE Transactions on Knowledge and Data Engineering*, 2019.
- [25] R. Herring, A. Hofleitner, S. Amin, T. Nasr, A. Khalek, P. Abbeel, and A. Bayen, "Using mobile phones to forecast arterial traffic through statistical learning," in *89th Transportation Research Board Annual Meeting*, 2010, pp. 10–14.
- [26] M. Li, A. Ahmed, and A. J. Smola, "Inferring movement trajectories from gps snippets," in *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, 2015, pp. 325–334.
- [27] J. Dai, B. Yang, C. Guo, C. S. Jensen, and J. Hu, "Path cost distribution estimation using trajectory data," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 85–96, 2016.
- [28] I. Jindal, X. Chen, M. Nokleby, J. Ye *et al.*, "A unified neural network approach for estimating travel time and distance for a taxi trip," *arXiv preprint arXiv:1710.04350*, 2017.
- [29] J. Hu, C. Guo, B. Yang, C. S. Jensen, and L. Chen, "Recurrent multi-graph neural networks for travel cost prediction," *arXiv preprint arXiv:1811.05157*, 2018.