A Distributed Travel Time Estimation Capability for Metropolitan-sized Road Transportation Networks

Arif Khan
ariful.khan@pnnl.gov
Pacific Northwest National Laboratory
Richland, Washington

Arun V Sathanur
arun.sathanur@pnnl.gov
Pacific Northwest National Laboratory
Seattle, Washington

Kelsey Maass
kmaass@uw.edu
University of Washington
Seattle, Washington

Robert Rallo
robert.rallo@pnnl.gov
Pacific Northwest National Laboratory
Richland, Washington

ABSTRACT

Street-level travel time estimation along with the temporal variations in patterns of travel times is an important component of traffic planning and operation in modern urban settings. In this work, we propose a scalable distributed-computing based methodology to leverage coarse-grained and aggregated travel time data to estimate the street-level travel times of a given metropolitan area. Our approach, termed *TranSEC* (Transportation State Estimation Capability), leverages easy-to-obtain, aggregated data sets with broad spatial coverage, such as the data published by Uber Movement and can handle road networks of very large size such as whole metropolitan areas. TranSEC is flexible enough to accommodate augmentation with fine-grained but potentially more expensive datasets, such as curated GPS-based data and probe data. Our proposed methodology uses a graph representation of the road network and combines several techniques such as weighted shortest-path routing, a trip sampling and a biased travel time sampling schemes, graph sparsification through betweenness centrality, and an iterative optimization flow that solves successive constrained least-squares optimization problems. TranSEC further uses graph partitioning tools to enable distributed solution to the problem. We demonstrate our method on the full Los Angeles metropolitan-area where aggregated travel time data is available for trips between traffic analysis zones using a 1280-core supercomputer and visualize the temporal traffic trends at the street-level.

KEYWORDS

Urban computing, Transportation, Graph algorithm, Constrained optimization

ACM Reference Format:
Arif Khan, Arun V Sathanur, Kelsey Maass, and Robert Rallo. 2020. A Distributed Travel Time Estimation Capability for Metropolitan-sized Road Transportation Networks. In *Proceedings of The 9th SIGKDD International Workshop on Urban Computing (UrbComp Workshop, ACM SIGKDD).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

UrbComp Workshop, ACM SIGKDD, August 2020, San Diego, CA, USA
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Big data and high performance computing paradigms are poised to revolutionize the transportation sector in the coming years [32]. A key requirement for a city transportation operation center is the global knowledge (global in this context refers to a large metropolitan area) of travel times and vehicle volumes which together constitute the traffic state on both arterial and freeway road segments.

However, in literature, interstate links have received high-levels of interest from the transportation research community as opposed to arterial segments even though both these classes of streets are important from the perspective of planning and operation. Potential reasons include wide-spread loop counter installations and availability of high quality data for inter-state links through public database access. Conversely, gathering arterial data is expensive due to the sheer number of probe sensors required for broad coverage [11].

Our work, termed, *TranSEC*, is concerned with filling the gap in arterial travel-time estimation though our framework doesn't distinguish between the types of links explicitly. This paper builds on two of our prior published works [17, 24] that focused on travel-time estimation at the TAZ (Traffic Analysis Zone) level and travel-time estimation at the street level, respectively. We start with the previously developed methodology and then proceed to describe the scaling methods and the distributed framework that makes it possible to estimate the travel times at scale, for large metropolitan road networks. We note that personal navigation devices and applications for smartphones, focus on optimizing travel times for individual drivers whereas *TranSEC* aims to provide a system-wide view at the street-level to the operators. Further, while transportation data companies offer products catering to these requirements, the cost of obtaining such highly granular data at scale can be very steep.

The main contributions of the full TranSEC workflow are listed as follows:

- We utilize the coarse-grained, aggregated TAZ-level data and the underlying road network fabric to setup an over-determined system of equations for street-level travel time estimation. The problem setup is made possible by a combination of weighted shortest path routing in graphs, trip and biased travel time sampling and a flexible scheme that

address pseudo-sparsification of the graph for better regularization and scalbility as well as provide the ability to augment the coarse TAZ-level data with accurate fine-grained data.
- We make use of constraints and the convex combinations of sequential iterates as a means of stabilizing solutions, improving convergence rates while avoiding explicit penalty functions for regularization. This allows us to make effective use of well-optimized least-squares routines for faster solution times.
- We design and implement a distributed strategy for metropolitan-area-wide problems based on spatial partitioning, solving independent problems and stitching the solutions for the cut links. We leverage the developed approach and produce travel-time estimates for different temporal windows for the Los Angeles metropolitan area.

The paper is organized as follows. Section 2 describes related work in the area of graph analytics, optimization, and machine learning applied to road networks. Section 3 describes the Uber Movement and the LA road network data used in this work. In the subsequent sections, we describe the forward model and simulated trips, followed by the optimization methodology (Section 4 and 5) used in this work. In Sections 6 and 7, we discuss scalable implementations of our proposed method in shared and distributed memory contexts respectively. Section 8 presents some insights into betweeness centrality and it's role in increasing the computational efficiency. We conclude the paper with a note on future work in Section 9.

2 RELATED WORK

Urban traffic modeling has seen a recent surge of interest in the use of graph analytics and machine learning methods. Reference [29] provides an overview of many of these methods. In this section we summarize the prior work related to both of these themes as applied to road transportation networks.

The authors in [25] use graph analytics on networks with three different weighting schemes to perform a statistical characterization of the Beijing road network. Many prior publications consider betweenness centrality [6] to be an important metric when applied to road networks, as it is argued to be a direct predictor of important links in urban transport. It has been shown that betweenness centrality is highly correlated with the traffic flow count on a road network [4, 12, 21], a natural result of including travel time as a factor when selecting trip routes. In real-world scenarios, however, route choices are also influenced by time-of-day and other socio-economic factors. Using these observations, the authors in [22] define an augmented betweenness centrality measure where shortest paths are weighted according to a traffic demand model based on census tracts and traffic analysis zones. The authors show that this new centrality measure correlates better with traffic flow than other centrality measures. Similarly, in [3] the authors employ analytics in the form of novel graph centrality measures to derive insights into the traffic flow patters in Singapore. Graph models, along with heterogeneous data sources, were leveraged to understand the urban traffic patterns in [19]. Finally, the authors in [8] utilize a grid-based fabric and cellular automata for modeling arterial traffic, resulting in gains in computational efficiency.

Many approaches make use of machine learning models and optimization methods to model various aspects of urban traffic flow. In [14], the authors leverage a deep-learning approach in the form of a diffusion convolutional recurrent neural network (DCRNN) to forecast short-term freeway traffic counts in the LA and San Francisco Bay Area networks. The authors in [2] also propose a deep-learning approach that brings together convolutional neural networks and recurrent neural networks with long short-term memory (LSTM) units, utilizing their architecture for short-term traffic count extrapolation at 349 locations on the Beijing road network. In a set of articles [23, 31], the authors leverage data from Bluetooth and GPS probe sensors for travel-time estimation and validation. Coupled hidden Markov models (CHMM) were used in [9] to model the evolution of traffic states, applied to a sparse taxi-fleet dataset for the San Francisco Bay area road network. In a subsequent publication [10] leveraging the same dataset, the authors employ a dynamic Bayesian network to learn arterial dynamics.

3 DATASETS

Our primary data sources for this work are from Uber Movement [27] and the road networks from OpenStreetMaps [20] for the Los Angeles metropolitan area. Uber has released a trove of aggregated and anonymized data on travel-time and average-speed statistics for a large number of cities around the world [27]. Uber Movement datasets [27] provide anonymized, aggregated, and coarse-grained O–D travel time statistics at the TAZ level for many metropolitan areas around the world. TAZs are small geographical units into which a given metropolitan area is divided, characterized by factors such as the total population, type of population, and employment. In this work, we focus on Uber's travel time data, which includes statistics for travel times between pairs of TAZs or census tracts for each hour of the day and day of the week. Similar datasets are also available through other sources (for specific cities) such as the New York City taxi-cab data [18].

Table 1 lists some basic network properties of the full road network, along with sub-networks formed with different radii around the downtown areas for Los Angeles. Other than the number of TAZs, most of the structural properties are similar across all of the networks. The number of TAZs included increases with the size of the graph for a given city network.

4 THE GRAPH-BASED FORWARD MODEL

In this section we describe a forward model for travel-time prediction, and in the following section we show how the parameters of this model can be estimated by means of an optimizer. Let $\mathcal{P}$ denote a set of edges that forms a path from vertex i to vertex j in the road network graph. The predicted travel time y_{ij} between the vertices i and j can be computed as

$$y_{ij} = \sum_{k \in \mathcal{P}} t_k = s^T t, \tag{1}$$

where t_k represents the expected travel time along edge k, the vector $t \in \mathbb{R}^M_{>0}$ represents the travel time along all M edges in the graph, and the binary vector $s \in \{0, 1\}^M$ encodes the edges in $\mathcal{P}$. The choice of optimal routing ($\mathcal{P}$) between the origin and destination vertices is a variable in the model. Routing is typically done with

Area	Vertices	Edges	TAZ	min(Deg)	max(Deg)	avg(Deg)
LA_DT+1	3,239	7,138	25	2	12	2.2
LA_DT+2	9,879	22,756	79	2	12	2.3
LA_DT+3	16,906	40,929	159	2	12	2.4
LA_metro	166,640	441,681	2,203	2	14	2.5

Table 1: Structural properties of Los Angeles road networks of varying sizes: one, two and three miles radii of downtown (DT) and the full networks.

special routing software such as the Open Source Routing Machine [16]. However, in this work, we use shortest-path routes where the edges are weighted by travel times [1, 15, 30]. Specifically, for initialization, we use weights determined by free-flow travel times $f \in \mathbb{R}^M_{>0}$, the time it takes to travel along a road segment free of congestion, computed by dividing the length of the road segment by the posted speed limit.

4.1 Training and testing tasks

Eq. 1 is based on O–D pairs in the road network graph. However, the Uber Movement data is provided at the much coarser granularity of TAZ O–D pairs, consisting of travel-time statistics computed over all trips originating at a given TAZ and ending in another during a particular hour of the day. Furthermore, not all TAZ pairs are included in the data.

For a given hour of the day and geographic area, we first collect all the TAZ O–D pair statistics available from Uber Movement. We then choose a random 90-10 split of the available TAZ O–D pairs for training and testing, respectively. For each of the TAZ O–D pairs, we simulate trips by sampling vertices from the origin and destination TAZs to form vertex O–D pairs, assigning trip times by sampling from a log-normal distribution based on the geometric mean and geometric standard deviation travel times given in the Uber dataset.

After estimating edge travel times using our vertex-level training data, testing is done at the TAZ level using the geometric mean travel time for all vertex O–D pairs present in a given test TAZ O–D pair. For example, suppose there are n_{ij} simulated trips from TAZ i to TAZ j. The estimated geometric mean travel time would then be computed as

$$g_{ij}(t) = \left(\prod_{k=1}^{n_{ij}} s_k^T t \right)^{\frac{1}{n_{ij}}}, \tag{2}$$

where vector $s_k \in \{0,1\}^M$ encodes the edges in the weighted shortest-path between sampled vertex O–D pair k. The vertex O–D sampling and travel-time sampling are described in detail below.

4.1.1 Vertex sampling. For each iteration k of our edge travel-time estimation algorithm, we sample N vertex O–D pairs each from our training and test sets, letting the number of simulated trips for each TAZ O–D pair be proportional to the size of the two TAZs. Specifically, for each TAZ O–D pair (i, j) in the current subset $\mathcal{U}_k$ of the Uber dataset, we let m_{ij} be the product of the number of vertices in origin TAZ i and destination TAZ j, then we set the number of simulated trips $n_{ij} = \text{int}\left(\left(\frac{m_{ij}}{\sum_{k,\ell \in \mathcal{U}_k} m_{k\ell}} \right) N \right)$.

We sample the n_{ij} origin and destination vertices for a given TAZ O–D pair uniformly from all vertices within their respective TAZ. We note that this process selects the shortest-path edges with a probability proportional to their local betweenness centrality, which in turn correlates with the importance of the edge with respect to traffic flow [4, 12, 21].

4.1.2 Biased Travel-time sampling. Sampling the trips and travel times independently can result in loss of correlation between trip lengths and sampled travel times. We therefore propose a simple heuristic to retain these correlations. Let $S \in \{0,1\}^{M \times N}$ represent the free-flow shortest-path matrix for our N simulated trips. We assign travel times $y \in \mathbb{R}^N_{>0}$ to trips based on the rank-ordering of free-flow shortest-path travel times $y_f = S^T f$ for all vertex O–D pairs within a TAZ O–D pair, summarized in Algorithm 1 below. Here $f \in \mathbb{R}^M_{>0}$ denotes the vector of free-flow travel times for each of the M edges.

Algorithm 1 Travel-time sampling
Input: Free-flow shortest-paths matrix S_k
Output: Sampled travel times vector y_k

```
1: for all TAZ O–D pairs $(i, j) \in \mathcal{U}_k$ do
2:     Sample $n_{ij}$ travel times $y_s$ from log-normal distribution
3:     Get indices $\ell_s$ and $\ell_f$ of longest to shortest travel times for $y_s$ and $y_t$, respectively
4:     for $m = 1, \ldots, n_{ij}$ do
5:         Assign $y_k(\ell_f(m)) = y_s(\ell_s(m))$
6:     end for
7: end for
```

In practice, our observations show that biased travel-time sampling significantly improve the results of travel time estimation.

For example, Fig. 1 illustrates the difference between estimated trip travel-time distributions with and without biased travel-time sampling. For 2000 trip times sampled from the log-normal distribution of a given TAZ O–D pair (left), the distribution of estimated trip travel times deviates less from the target distribution (red line, same across panels) when we bias the travel times assigned to each vertex O–D pair (center) than when we do not (right).

5 THE OPTIMIZATION PROCESS

Our approach solves a series of constrained least-squares problems to fit edge travel times to simulated trips (vertex O–D pairs, travel times, and routes) with travel-time statistics consistent with the Uber dataset. For each iteration k, we sample N vertex O–D pairs with sampled travel times vector $y_k \in \mathbb{R}^N_{>0}$ and weighted shortest-paths matrix $S_k \in \{0,1\}^{M \times N}$. Our goal is to then estimate edge

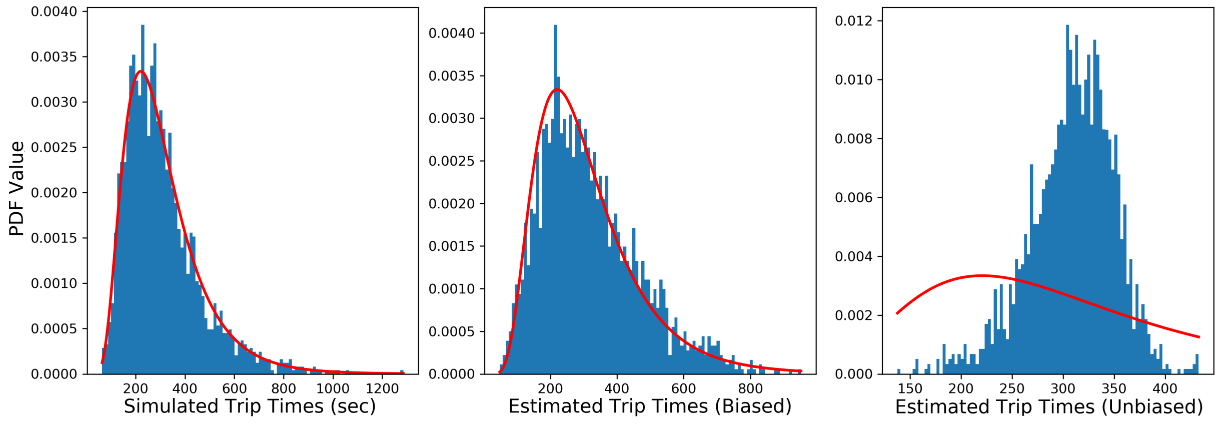

Figure 1: Illustrating the effect of biasing travel-time assignments for simulated trips for a TAZ O-D pair with $n_{ij} = 2000$ trips. (Left) Distribution of simulated travel times y_s according to the log-normal distribution drawn in red. (Center) Distribution of estimated trip travel times $S^T t \approx y$ computed using biased sampling $y(\ell_f(k)) = y_s(\ell_s(k))$. (Right) Distribution of estimated trip travel times $S^T t \approx y$ computed using unbiased sampling $y = y_s$.

travel times vector $t \in \mathbb{R}^M_{>0}$ so that $S_k^T t \approx y_k$ to satisfy our forward model in Eq. 1. Thus we have a system of N equations with M unknowns where M is the number of road segments (or edges in the graph). The estimates are then updated using a convex combination of the previous estimates and the solution found by minimizing the mean-squared error with constraints on the unknown coefficients, illustrated by Eqs. 3-5 below.

$$\hat{t} = \underset{\alpha g_k \le t \le \beta h_k}{\arg\min} \left\| S_k^T t - y_k \right\|_2^2 \tag{3}$$

$$t_{k+1} = (1 - \lambda_k) t_k + \lambda_k \hat{t} \tag{4}$$

$$\lambda_{k+1} = \theta_k \lambda_k \tag{5}$$

In our implementation, we initialize $t_0 = f$ with the free-flow estimates and let $\lambda_0 = 1$, we constrain the elements of the vector $\hat{t}$ to be bounded below by $\alpha g_k = 0.8 f$ and above by $\beta h_k = 1.25 t_k$, and we update our weight λ_k using the constant $\theta_k = 0.9$ for all k. Our weighted shortest-paths matrix S_k is recomputed each iteration using our current estimates t_k as weights, and our constrained least-squares sub-problems are solved using the `lsq_linear` function from the well-known Python scientific computing library SciPy[28]. We run the optimizer until the estimates converge (measured by the magnitude of the average change in the solution vector between iterations, Eq. 6 with $\delta = 0.01$), or we reach a maximum number of iterations $k_{\max}$. This iterative scheme is similar to the one proposed in [1] but with a choice of optimizer that favors scalability in conjunction with a number of heuristics that improve convergence. The workflow of our proposed travel-time estimation algorithm is given in Algorithm 2.

$$\frac{1}{M} \|t_k - t_{k-1}\|_2 \le \delta \tag{6}$$

We implement the workflow Algorithm 2 using Python (*v3.7*). Fig. 2 shows the convergence of the travel-time error for LA Downtown (both training and test sets) for the road network at 3am and 6pm, measured by

$$\epsilon_k = \sqrt{\frac{1}{N} \sum_{(i,j) \in \mathcal{U}_k} n_{ij} \left(\log g_{ij}(t_k) - \log G_{ij} \right)^2}, \tag{7}$$

where N is the total number of trips in the training (test) data, n_{ij} is the number of trips for TAZ O–D pair $(i, j) \in \mathcal{U}_k$ in the training (test) subset, $g_{ij}(t_k)$ is the current estimated geometric mean travel time (Eq. 2), and G_{ij} is the geometric mean travel-time from the Uber Movement dataset. This RMSLE error metric represents the relative mean-squared error in the logarithm of the estimated geometric mean travel time and is known to be a better metric for error estimation in high variance situations [1]. Further, we note that, low values of RMSLE are equivalent to fractional errors since $\exp(x) \approx (1 + x)$, for small values of x (say $x \le 0.25$ and exp denotes the base of the natural logarithm.

Algorithm 2 Travel-time Estimation
Input: Travel time statistics for TAZ O–D pairs; Road network $G = (V, E)$; Number of trips N
Output: Estimated travel times along edges t

```
1: Determine number of trips $n_{ij}$ for each TAZ O–D pair (Section 4.1.1)
2: Initialize edges with free-flow travel times $t_0 = f$ and $k = 0$
3: while Eq. 6 not satisfied do
4:     Sample $N$ vertex O–D pairs
5:     Compute shortest-path matrix $S_k$ using weights $t_k$
6:     Compute $y_k$ using biased travel-time sampling (Algorithm 1)
7:     Compute $t_{k+1}$ according to Eqs. 3-5
8:     $k = k + 1$
9: end while
```

6 SCALING TRANSEC FOR METROPOLITAN-SIZED ROAD NETWORKS

In this section, we discuss strategies to scale TRANSEC for travel time estimation in large metropolitan area networks. Referring to Table 1, the LA metropolitan road network consists of roughly $166K$

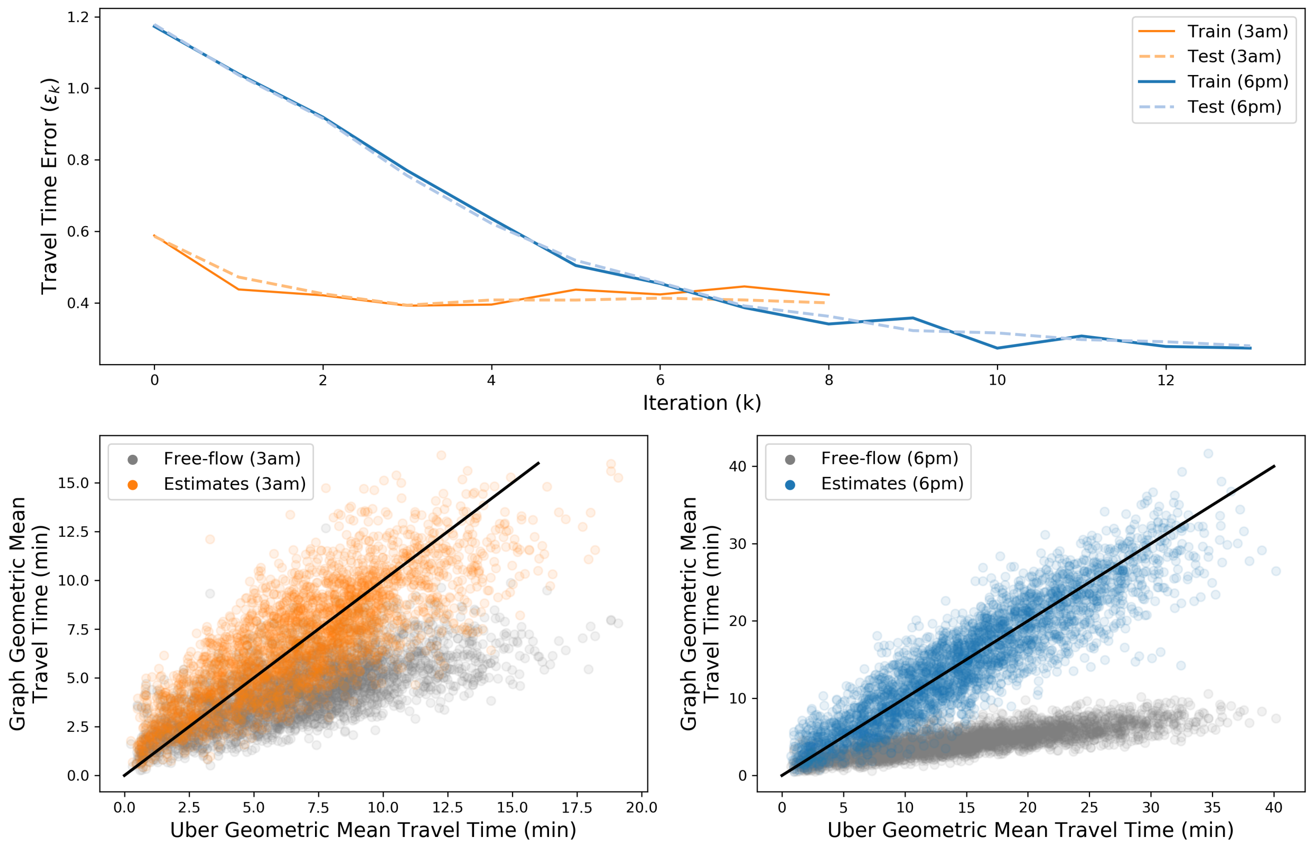

Figure 2: Travel-time estimation results for downtown LA for off-peak (3am) and peak traffic (6pm). Top: Convergence of travel time error ϵ_k. Bottom: Geometric mean travel times g_{ij} calculated using free-flow edge weights and estimated edge weights compared to ground-truth data G_{ij} from Uber.

vertices (intersections) and $441K$ edges (segments). The computational complexity of our proposed model is proportional to the number of trips N and the number of the edges M, i.e., $O(NM)$. The over-determined systems of equations that we setup, usually sets the number of trips to be proportional to the number of unknowns ($N \approx 1.2M$). Thus the complexity of the solution process is $O(M^2)$. Due to the large problem size, for example, $441K$ edges in the LA metropolitan road network and approximately $550K$ trips, estimating travel times for metropolitan scale network becomes prohibitively large with respect to compute time as well as memory requirement. Our goal is to design a flexible scheme, simple enough so that it can solve large problems on a single computer as well as highly efficient that can run on multiple machines, if available, in parallel in order to reduce the compute time. We achieve these goals by employing a three step strategy as follows: i) partition the problem into (potentially) independent sub-problems, ii) solve each of these sub-problems independently and ii) finally, stitch the sub-solutions in order to obtain final travel time estimates. We have already discussed the second step in previous sections which is essentially the Algorithm 2. We will discuss partitioning and stitching in details in the following subsections.

6.1 Partitioning the problem

We partition the problem along the spatial dimensions, that is, given a road network, we partition the network into mutually exclusive geographic parts. There are two major challenges for this step. First, we need to minimize the cut edges, i.e., the road segments that straddle two neighboring partitions. The travel times on these segments are not solved directly in the second step and so we infer travel times on these edges in the last step while stitching the sub-solutions. Therefore, minimizing the number of cut edges reduces the inference error in the estimate. Second, we need to keep the size of each partition balanced as much as possible in order to achieve parallel efficiency. We use popular graph partitioning algorithm *METIS* [13] which addresses both of these issues.

The computational speed up using spatial partitioning scheme is linear in the number of the partitions on a single machine. If we partition a problem into k equally sized sub-problems. Each sub-problem will take $\frac{1}{k^2}$ of the total time since solver complexity is quadratic. Therefore, the total time using partition scheme would be $k \times \frac{1}{k^2}$ of the original (un-partitioned) compute time, resulting a k fold speed-up.

6.2 Stitching the sub-solutions

The final step is to assemble all the sub-solutions and estimate the travel time on the cut-edges. We employ a simple heuristic of (approximate) speed continuity for this purpose. For each cut edge, we find the preceding and the following road segments, in the same geographical direction with their associated estimated travel times. We compute the average speed from those travel times and use it to compute how long it would take to travel along the cut road segment on average.

Algorithm 3 Stitching the sub-solutions
Input: Sub-Solutions sub_res_i; Sub-problems $partitions$,Road network $G = (V, E)$;
Output: Estimated travel times along edges t

```
1: Determine the cut edges $E_c$ from the $G$ and $partitions$
2: for each cut edge, $e_i \in E_c$ do
3:     $t_{prev}$ = travel time of previous road segment of $e_i$
4:     $t_{next}$ = travel time of following road segment of $e_i$
5:     $s_{prev}$ = speed of road segment using $t_{prev}$
6:     $s_{next}$ = speed of road segment using $t_{next}$
7:     $s_i = average(s_{prev}, s_{next})$
8:     $t_i$ = travel time using $s_i$
9: end for
```

Algorithm 4 describes the scalable TRANSEC framework which can estimate travel times of large metropolitan sized road network with the three steps discussed in this section.

Algorithm 4 TRANSEC
Input: Travel time statistics for TAZ O–D pairs; Road network $G = (V, E)$; Number of trips N, Number of Partitions, P
Output: Estimated travel times along edges t

```
1: //Step 1: Partition the road network and send it to remote computers
2: $partitions = METIS(G, P)$
3:
4: //Step 2: Travel time estimation
5: $temp\_res = \emptyset$
6: for each $p_i \in partitions$ do
7:     Create subgraph,$G_p = induce(G, p_i)$
8:     $sub\_res_i = Estimate\_travel\_time(G_p, OD, N)$        ▷ Algorithm 2
9:     $temp\_res = temp\_res \cup sub\_res_i$
10: end for
11:
12: //Step 3: Collect sub-solutions and Stitching the results
13: $res = Stich(temp\_res, G, partitions)$        ▷ Algorithm 3
14: return $res$
```

7 DISTRIBUTED MEMORY TRANSEC

In this section we discuss, the distributed memory algorithm of the TRANSEC in Algorithm 5. We implement the algorithm using Python (*v3.7*) along with Python specific MPI implementation, *MPI4PY* [5] for the inter node (computer) communication and synchronization. The distributed memory TRANSEC algorithm is shown in Algorithm 5.

Without the loss of generality, we assume that the number of available compute nodes is equal to the number of partitions so that each node is responsible for estimating travel times for exactly one partition. If the number of compute nodes is less than that of partitions, then some compute nodes will be assigned to more than one partition which in turn can be solved in serial using Algorithm 4. Algorithm 5 starts by creating standard MPI library communicators for managing communication and synchronization among the computers. The master node (rank 0) then partitions the network and sends partition information to the corresponding client nodes (line 4-8). After receiving the partition information (line 10) from the master node, each node (including the master) computes the travel times for the road segments independently for the assigned partition (line 14-15). When the estimation is over, each client sends the result back to the master (line 25). Upon receiving all the sub-solutions, the master node stitches the sub-solutions and obtains the final travel time estimate (line 18-24).

It is evident from the discussion that increasing the number of partitions (as well as compute nodes) speeds up the computation process. However, more partitions mean more cut edges which may have negative impact on the solution quality. We experimented with this trade-off and concluded that using 64 partition for LA metropolitan area gives us a good quality solutions with efficient speed up in computation. For the LA metro area network, with $167k$ intersections and $441k$ road segments, the serial run with 64 partitions takes 32 hours and 27 mins. The distributed memory flow

Algorithm 5 Distributed Memory TRANSEC
Input: Travel time statistics for TAZ O–D pairs; Road network $G = (V, E)$; Number of trips N, Number of Partitions, P
Output: Estimated travel times along edges t

```
1:
2: comm=Create MPI_communicators for $P$ computers
3:
4: //Step 1: Partition the road network and send it to remote computers
5: if $my\_rank == 0$ then
6:     $partitions = METIS(G, P)$
7:     for each $p_i \in partitions$ do
8:         $MPI\_Send(p_i, i, comm)$
9:     end for
10: else
11:     $MPI\_Recv(p_i, 0, comm)$
12: end if
13:
14: //Step 2: Travel time estimation
15: Create subgraph,$G_p = induce(G, p_i)$
16: $sub\_res_i = Estimate\_travel\_time(G_p, OD, N)$        ▷ Algorithm 2
17:
18: //Step 3: Collect sub-solutions and Stitching the results
19: if $my\_rank == 0$ then
20:     for each computer, $i$ do
21:         $MPI\_Recv(sub\_res_i, i, comm)$
22:         $temp\_res = temp\_res \cup sub\_res_i$
23:     end for
24:     $res = Stitch(temP\_res, G, partitions)$        ▷ Algorithm 3
25: else
26:     $MPI\_Send(sub\_res_i, 0, comm)$
27: end if
28:
29: return $res$
```

is completed in about 37 mins using 64 computing nodes giving us 52× speedup and a 0.82 parallel efficiency.

The parallel efficiency is defined as the ratio of speed up over the number of compute nodes. The parallel efficiency is 1 means the algorithm has perfect linear strong scaling performance. However due to the overhead of inter node communication and synchronization, we achieve parallel efficiency 0.82 which is a reasonably efficient scaling performance by our proposed algorithm. We show that the distributed TranSEC scales up to 64 compute node each with 20 Intel E5-2680, 2.80GHz cores (i.e., 1280 cores in total) with 52× speed up for LA metropolitan area problem.

We further note that, the distributed algorithm allows us to compute the solutions to the independent partial problems in parallel. In the absence of a cluster or a supercomputer, the independent problems can be solved serially and the stitching algorithm applied after all the partial solutions are obtained.

Next, we discuss the solution quality in Figure 3 for LA metropolitan area from 2 PM to 7 PM in order to see how traffic pattern changes from off peak to peak and back to off peak traffic. We plot the excess travel time (estimated (t) - free flow (f)) as a percent of free flow travel time in the LA metropolitan area. It captures how slow a vehicle moving w.r.t. the free flow (no congestion) scenario. We consider a road segment is severely congested if the excess travel time is equal to or above 80% over the free flow, i.e., roughly driving 17 mph in a 30 mph zone. We observe that from 2 PM onward the congestion increases steadily until 5 PM then eases out gradually at 7 PM which aligns with real world observation. The figure also identifies known hotspots such as LA downtown, Universal Studio and Pasadena areas as well.

8 GRAPH PSEUDO-SPARSIFICATION

In this section, we discuss strategies to further speed up the computation while also assisting in the regularization of the system of equations. Ensuring accuracy of the travel time estimation on heavily utilized arterial and highway segments is very important. This means that we can potentially reduce the number of unknowns by removing the least-utilized segments from the system. These are typically links that have very low edge-betweenness values. However, a straight-forward strategy to drop the low betweenness edges will not work in practice. This can lead to connectivity issues as well as skew the routing and hence the travel time estimation.

For example, consider two paths between vertex u and v: the shortest route through intermediate vertices s and t, and an alternative, but longer, route through vertices x and y. If a sparsification algorithm removes the road segment between s and t based on some metric (e.g., low edge betweenness centrality), our model would incorrectly estimate the travel time from u to v using the available road segment through x and y.

In order to address the above issues, our approach is to pseudo-sparsify the underlying road network. Rather than remove edges from the graph, we sort road segments into two sets based on their significance with respect to traffic flow (e.g., betweenness centrality). We then estimate the travel-time along the edges with the top $p\%$ significance, setting the travel times along the remaining edges to the free-flow travel times given that low betweenness typically translates to less traffic volume as well. In our implementation, we sort edges based on their betweenness centrality, computed using shortest-paths weighted by free-flow travel times described in Algorithm. For instance, if i are the indices of the edges with the top $p\%$ betweenness and j are indices of the remaining edges, we solve a modified version of Eqs. 3-4 each iteration:

$$\hat{t} = \underset{\alpha g_k(i) \leq t \leq \beta h_k(i)}{\arg\min} \left\| S_k(i)^T t + S_k(j)^T f(j) - y_k \right\|_2^2 \quad (8)$$

$$t_{k+1}(i) = (1 - \lambda_k) t_k(i) + \lambda_k \hat{t}, \quad t_{k+1}(j) = f(j) \quad (9)$$

Note that this approach can provide for modest speed-ups in the computation flow. Choosing the top 70% links leads to a $2X$ speed-up. However, a more important side-effect of this approach is to remove the unknowns that appear in a very few equations because of their low betweenness. Note that the action of sampling trips and the associated shortest-paths is very similar to estimating betweenness via a sampling approach [7]. This will lead to a better least-squares estimate in a manner similar to setting the coefficients of low-importance variables in Lasso regression[26] to zero.

We test our pseudo-sparsification approach using different values of p (Fig. 4). In Fig. 4 (left), we first run our algorithm on the full graph (i.e., $p = 100\%$), and plot the relative difference between our estimated travel times and the initial free-flow travel times $|t - f|/f$ versus the percentile of the edges sorted by betweenness centrality. We observe that roughly 15% of lowest betweenness edges retain their free-flow travel time as the final estimate, suggesting that we can set these edges to their free-flow travel times to reduce the number of unknowns.

Of course, drivers rarely maintain the exact speed limit. To account for this uncertainty in free-flow travel times, we sort the edges into bins based on their betweenness percentiles and plot the fraction of edges within each bin that have a relative difference within a factor of $q\%$ of their free-flow travel times (Fig. 4, right). We observe that the high betweenness edges are less likely to have an estimated travel time close to their free-flow times irrespective of the level of uncertainty q. Therefore, we can pseudo-sparsify the graph to different extents depending upon the level of uncertainty that can be tolerated.

9 CONCLUSIONS AND FUTURE WORK

In this work, we consider the problem of estimation of travel times at the street-level in a metropolitan-sized road network. Our starting point for the data is the road network and incomplete TAZ-TAZ travel time aggregate statistics. The solution process is aided by sampling trips from the training data and solving for the street-level travel times through an iterative process that uses constrained least squares optimization at each step.

We further present a scaling strategy based on graph-partitioning and implement the same on distributed memory machines to enable solving problem sizes as large as the Los Angeles metropolitan area network with $441K$ road segments in 37 minutes using a 64-node, 1280-core supercomputer. Finally, we present a pseudo-sparsification strategy that can lead to additional speedups while assisting regularizing the system of equations.

Our future work is focused on better stitching strategies, uncertainty quantification and further validation of the methodology on other geographical areas.

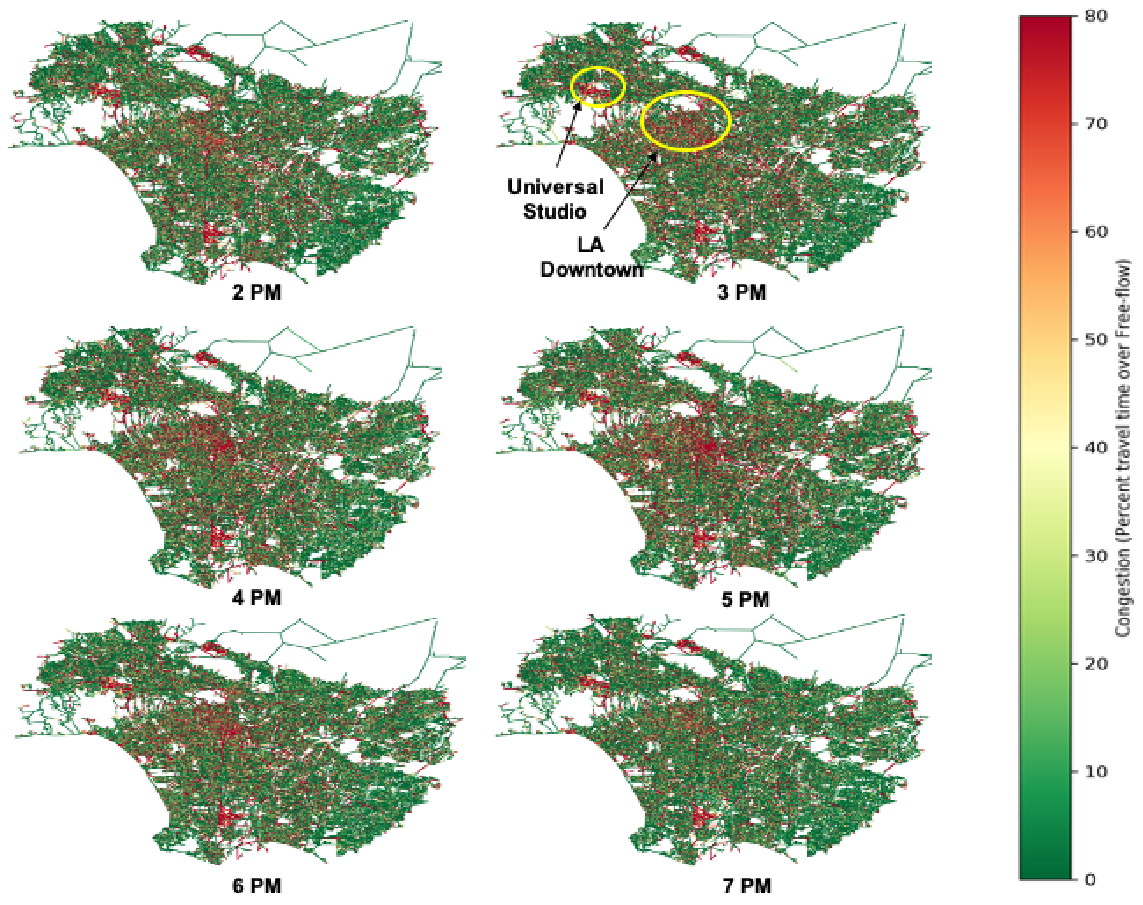

Figure 3: Estimated excess travel time ($t - f$) as a percent of free-flow travel time f in LA metropolitan area.

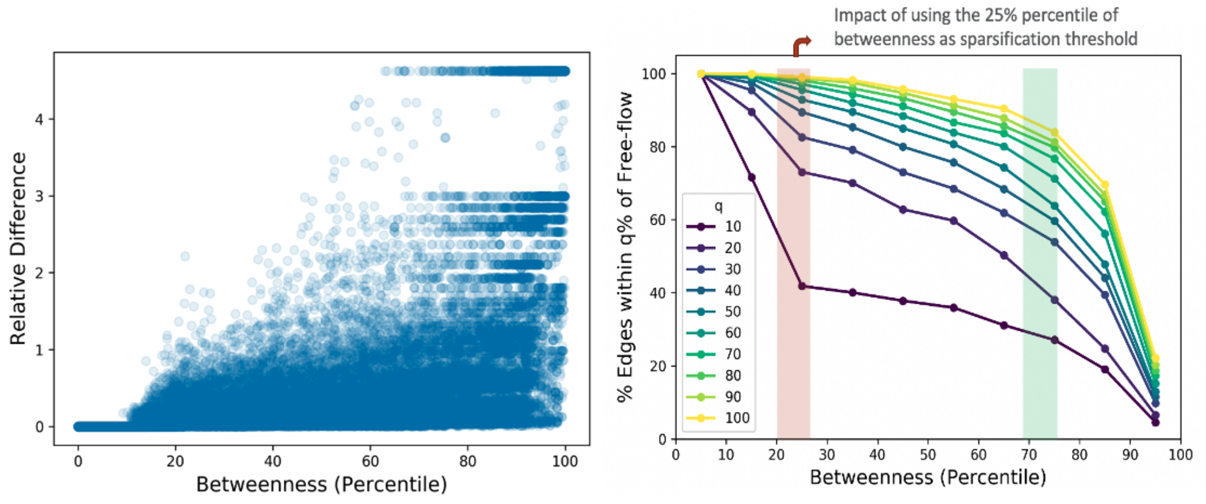

Figure 4: Relative difference ($|t - f|/f$) between estimated travel time t and free-flow travel time f by percentile of betweenness for peak downtown LA traffic (left) and fraction of estimated edge travel times that are within $q\%$ of free-flow ($|t - f|/f \leq q\%$), where edges are binned by percentile (right). Edge betweenness centrality is calculated using shortest paths weighted by free-flow travel time.

ACKNOWLEDGMENTS

This report and the work described were sponsored by the U.S. Department of Energy (DOE) Vehicle Technologies Office (VTO) under the Big Data Solutions for Mobility Program, an initiative of the Energy Efficient Mobility Systems (EEMS) Program. The following DOE Office of Energy Efficiency and Renewable Energy (EERE) managers played important roles in establishing the project concept, advancing implementation, and providing ongoing guidance: Prasad Gupte and David Anderson.

REFERENCES

[1] Dimitris Bertsimas, Arthur Delarue, Patrick Jaillet, and Sébastien Martin. 2019. Travel Time Estimation in the Age of Big Data. *Operations Research* 67, 2 (2019), 498–515.

[2] Xingyi Cheng, Ruiqing Zhang, Jie Zhou, and Wei Xu. 2018. Deeptransport: Learning spatial-temporal dependency for traffic condition forecasting. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[3] Yew-Yih Cheng, Roy Ka-Wei Lee, Ee-Peng Lim, and Feida Zhu. 2015. Measuring centralities for transportation networks beyond structures. In *Applications of social media and social network analysis*. Springer, 23–39.

[4] Paolo Crucitti, Vito Latora, and Sergio Porta. 2006. Centrality in networks of urban streets. *Chaos: an interdisciplinary journal of nonlinear science* 16, 1 (2006), 015113.

[5] Lisandro D Dalcin, Rodrigo R Paz, Pablo A Kler, and Alejandro Cosimo. 2011. Parallel distributed computing using Python. *Advances in Water Resources* 34, 9 (2011), 1124–1139.

[6] Linton C Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* (1977), 35–41.

[7] Robert Geisberger, Peter Sanders, and Dominik Schultes. 2008. Better approximation of betweenness centrality. In *2008 Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 90–100.

[8] PJ Gundaliya, Tom V Mathew, and Sunder Lall Dhingra. 2008. Heterogeneous traffic flow modelling for an arterial using grid based approach. *Journal of Advanced Transportation* 42, 4 (2008), 467–491.

[9] Ryan Herring, Aude Hofleitner, Pieter Abbeel, and Alexandre Bayen. 2010. Estimating arterial traffic conditions using sparse probe data. In *13th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 929–936.

[10] Aude Hofleitner, Ryan Herring, Pieter Abbeel, and Alexandre Bayen. 2012. Learning the dynamics of arterial traffic from probe data using a dynamic Bayesian network. *IEEE Transactions on Intelligent Transportation Systems* 13, 4 (2012), 1679–1693.

[11] Timothy Hunter, Ryan Herring, Pieter Abbeel, and Alexandre Bayen. 2009. Path and travel time inference from GPS probe vehicle data. *NIPS Analyzing Networks and Learning with Graphs* 12, 1 (2009), 2.

[12] Bin Jiang. 2009. Street hierarchies: a minority of streets account for a majority of traffic flow. *International Journal of Geographical Information Science* 23, 8 (2009), 1033–1048.

[13] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.

[14] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJiHXGWAZ

[15] Eric Hsueh-Chan Lu, Chia-Ching Lin, and Vincent S Tseng. 2008. Mining the shortest path within a travel time constraint in road network environments. In *2008 11th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 593–598.

[16] Dennis Luxen and Christian Vetter. 2011. Real-time routing with OpenStreetMap data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (Chicago, Illinois) *(GIS '11)*. ACM, New York, NY, USA, 513–516. https://doi.org/10.1145/2093973.2094062

[17] Kelsey Maass, Arun V Sathanur, Arif Khan, and Robert Rallo. 2020. Street-level Travel-time Estimation via Aggregated Uber Data. In *2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing* (Seattle, WA, USA). 76–84.

[18] NYC Taxi and Limousine Commission. 2019. NYC yellow and green taxi trip records, . https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

[19] Kamaldeep Singh Oberoi, Géraldine Del Mondo, Yohan Dupuis, and Pascal Vasseur. 2017. Spatial Modeling of Urban Road Traffic Using Graph Theory. In *Spatial Analysis and GEOmatics 2017*.

[20] OpenStreetMap contributors. 2019. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org.

[21] Sergio Porta, Paolo Crucitti, and Vito Latora. 2006. The network analysis of urban streets: a primal approach. *Environment and Planning B: planning and design* 33, 5 (2006), 705–725.

[22] Rami Puzis, Yaniv Altshuler, Yuval Elovici, Shlomo Bekhor, Yoram Shiftan, and Alex Pentland. 2013. Augmented betweenness centrality for environmentally aware traffic monitoring in transportation networks. *Journal of Intelligent Transportation Systems* 17, 1 (2013), 91–105.

[23] Hesham A Rakha, Hao Chen, Ali Haghani, Xuechi Zhang, Masoud Hamedi, et al. 2015. *Use of Probe Data for Arterial Roadway Travel Time Estimation and Freeway Medium-term Travel Time Prediction.* Technical Report. Mid-Atlantic Universities Transportation Center.

[24] Arun V. Sathanur, Vinay Amatya, Arif Khan, Robert Rallo, and Kelsey Maass. 2019. Graph Analytics and Optimization Methods for Insights from the Uber Movement Data. In *Proceedings of the 2Nd ACM/EIGSCC Symposium on Smart Cities and Communities* (Portland, OR, USA) *(SCC '19)*. ACM, 2:1–2:7.

[25] Zhao Tian, Limin Jia, Honghui Dong, Fei Su, and Zundong Zhang. 2016. Analysis of urban road traffic network based on complex network. *Procedia engineering* 137 (2016), 537–546.

[26] Robert Tibshirani. 1997. The lasso method for variable selection in the Cox model. *Statistics in medicine* 16, 4 (1997), 385–395.

[27] Uber Technologies, Inc. 2019. Data retrieved from Uber Movement, (c) 2019 . https://movement.uber.com.

[28] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods* 17, 3 (2020), 261–272.

[29] Eleni I Vlahogianni, Matthew G Karlaftis, and John C Golias. 2014. Short-term traffic forecasting: Where we are and where we're going. *Transportation Research Part C: Emerging Technologies* 43 (2014), 3–19.

[30] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. 2012. Shortest path and distance queries on road networks: An experimental evaluation. *Proceedings of the VLDB Endowment* 5, 5 (2012), 406–417.

[31] Xuechi Zhang, Masoud Hamedi, and Ali Haghani. 2015. Arterial travel time validation and augmentation with two independent data sources. *Transportation Research Record* 2526, 1 (2015), 79–89.

[32] Li Zhu, Fei Richard Yu, Yige Wang, Bin Ning, and Tao Tang. 2018. Big data analytics in intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems* 20, 1 (2018), 383–398.