# Think Like A Graph: Real-Time Traffic Estimation at City-Scale

Zhidan Liu, *Member, IEEE*, Pengfei Zhou, Zhenjiang Li, *Member, IEEE*, and Mo Li, *Member, IEEE*

**Abstract**—This paper presents a graph processing based traffic estimation system, GPTE, which is able to achieve high accuracy and high scalability to support city scale traffic estimation. GPTE benefits from its non-linear traffic correlation modeling and the graph-parallel processing framework built on clustered machines. By representing the road network as a property graph, GPTE decomposes the numerous computations involved in non-linear models to vertices and performs traffic estimation via neural network modeling and iterative information propagation. This paper presents our experiences in designing and implementing GPTE on top of the Spark, an emerging cluster computing framework. Extensive experiments are performed with real-world data input from Singapore's transport authority. Experimental results show that GPTE achieves as high as 88 percent accuracy in traffic estimation and up to $8\times$ performance gain in computation efficiency with the optimization techniques applied. Comparison study demonstrates that GPTE outperforms the baseline solutions by 34 percent on accuracy and 46 percent on processing time.

**Index Terms**—Traffic estimation, graph-parallel processing, non-linear correlation modeling

## 1 Introduction

Accurate and timely traffic information is of essential importance to urban transportation, and tremendous efforts have been put in efficiently monitoring the traffic conditions in the past decades. Conventional methods rely on deploying intrusive sensing infrastructures, e.g., traffic cameras or inductive loop detectors [43], to actively detect traffic conditions. Due to the excessive deployment and maintenance overheads, it becomes prohibitive when adopting such intrusive solutions at city scale, and as a result the coverage is limited to certain busy road segments or junctions in most cities. Many recent studies resort to data-driven solutions, where location reports collected from driving vehicles on roads are leveraged to derive traffic conditions [23], [41], [52]. Vehicles, equipped with GPS devices, can periodically report their instant status including locations, travel speeds, travel directions, etc. This information can be used to estimate instant traffic speeds of roads covered by probe vehicles. Such passive traffic sensing methods avoid expensive infrastructure deployment and largely extend the coverage of traffic estimation. As an example of practice, the Land Transport Authority (LTA) [1] of Singapore is currently making use of the GPS reports from 12000+ taxi fleet to derive the city wide road traffic in its TrafficScan portal that publishes the live traffic information to all Singaporean citizens [2], [9]. Despite these advantages, such solutions are inherently limited by the number of probe vehicles. In TrafficScan, even with 12000+ taxis the available GPS reports (which are generated every 30 seconds in each 15 mins time slot) are still sparse when compared with the 58000+ road segments of the city. The Google Traffic,[1] with its combined usage of traffic data acquired from local transport authority (e.g., LTA in the case of Singapore) and crowdsourced location reports from mobile phones, shares similar limit in data availability and coverage.

Many works were proposed to address the data sparsity issue in passive traffic estimation. Most of them exploit the traffic correlations among different road segments to recover the complete traffic from incomplete road measurements [23], [40], [52]. The traffic correlations used in those works are explicitly or implicitly modeled linearly mainly to reduce computations for large road networks. The practical traffic, however, is influenced by various factors, e.g., intricate road network, transport regulations, mixed traffic flows, etc., and thus are much more complex than linearly, which requires advanced modeling for more accurate estimation. There exist a few studies that consider non-linear traffic correlation models [4], [16], [40], all of which, however, suffer from poor scalability due to enormous computation overheads involved in non-linear models, which strictly limits their practical applicability to larger scale traffic estimation.

This paper proposes a non-linear model to characterize the traffic correlations and derive traffic with the 12000+ taxi probes from LTA. In order to cope with the heavy computations introduced by the model, we develop a graph processing framework that can be efficiently executed in parallel on computer cluster. In the proposed framework, the road network is represented as a property graph, where vertices are road segments and edges are formed between


- *Z. Liu is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China. E-mail: liuzhidan@szu.edu.cn.*
- *P. Zhou is with the School of Software, Tsinghua University, Beijing 100084, China. E-mail: zhoupf05@tsinghua.edu.cn.*
- *Z. Li is with the Department of Computer Science, City University of Hong Kong, Hong Kong. E-mail: zhenjiang.li@cityu.edu.hk.*
- *M. Li is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798. E-mail: limo@ntu.edu.sg.*


*Manuscript received 5 Apr. 2018; revised 25 Aug. 2018; accepted 19 Sept. 2018. Date of publication 4 Oct. 2018; date of current version 28 Aug. 2019. (Corresponding author: Zhidan Liu.)*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TMC.2018.2873642*

1. The details how Google Traffic derives live traffic are not public.

connected road segments. By distributing the property graph among machines, we decompose all computation tasks at each vertex and perform complete traffic estimation via information propagation among vertices. This solution embraces non-linear traffic correlations for higher estimation accuracy and can be highly parallelized for city-scale traffic estimation.

The idea being attractive, developing a practical system out of it is challenging for at least two reasons. First, it is non-trivial to model non-linear traffic correlations within a graph. As taxis freely travel across roads, the traffic states of graph vertices are randomly sampled and they form a time-evolving graph. Due to such dynamics, the vertices with known traffic states are changing over time. As a result, it is impossible to maintain a fixed correlation model throughout traffic estimations. Second, a number of system details need to be carefully addressed when implementing the full system. In particular, when we fit our solution into a cluster computing platform, we must not only consider computation cost but also minimize the communication overhead associated with data exchange and computing threads running on different machines. This requires wise treatments to the traffic data given road network structure and characteristics.

We propose *G*raph-parallel *P*rocessing based *T*raffic *E*stimation - GPTE, which addresses above challenges. GPTE represents road network data as a property graph and annotates vertex states with real-time traffic samplings. GPTE builds artificial neural network (ANN) models to capture the correlations and iteratively propagates traffic information from annotated vertices to those vertices of unknown states. To deal with dynamics in the time-evolving graph, for each unannotated vertex GPTE dynamically selects correlated vertices from its annotated neighbors, and builds an instant ANN model to infer its traffic state. We build GPTE based on the latest cluster computing framework Spark [44], and make use of interfaces provided by the recent graph processing engine GraphX [14]. To reduce communication cost during the cluster execution, we propose a geography-aware graph partitioner that optimizes the data layout on different machines. We improve the efficiency of information propagation among vertices using multi-hop message broadcast scheme and redundant message elimination. In addition, we incorporate a set of optimization techniques to improve the accuracy and efficiency of data processing.

To the best of our knowledge, this is the first real-time traffic estimation solution that incorporates advanced traffic correlation modeling for large scale road networks. We systematically evaluate the performance of GPTE with real-world traffic data provided by LTA. The experimental results show that in average GPTE can accomplish the traffic estimation for the entire city in every 15 mins time slot in 34 seconds and achieve as high as 88 percent estimation accuracy.

In the rest of this paper, we present the motivation and design in Sections 2 and 3, respectively. The system implementation is detailed in Section 4. Optimizations are presented in Section 5. The evaluations are described in Section 6. We review related works in Section 7. Finally Section 8 concludes this paper.

## 2 Traffic Estimation with Probes

*The Problem.* The road network of an urban city is composed of a number of roads of different types, e.g., expressways, major roads, minor roads, etc. Each road is further divided into smaller road segments for better granularity in traffic estimation. The traffic condition of a road segment $r_i$ can be measured by the average traffic speed $v_i$ within a time slot, which used to be collected via expensive sensing infrastructures, e.g., inductive loop detectors [43], at a few important road segments. An appealing alternative that has been recently practiced is data-driven and based on the traffic samplings from probe vehicles. A typical traffic sampling contains a timestamp, location, travel speed, travel direction, etc. [38]. For each road segment $r_i$, its traffic condition $v_i$ can be approximated as the average travel speed of all probe vehicles passing by [40], [52]. The approximation is considered credible if the road segment is sampled by a sufficient number, e.g., $\geq \lambda$, of probe vehicles [39]. The objective of traffic estimation is to derive the *timely*, *accurate*, and *complete* traffic conditions for all road segments based on the available traffic samplings from probe vehicles.

A concrete practice is TrafficScan [2] that has been developed and used by the LTA of Singapore. Since 1999, LTA has been trying to exploit the traffic samplings collected from driving taxis to conduct traffic estimation and provided such information to the public for their route planning. More than 12000 taxis have recently been engaged in providing traffic samplings and the LTA is planning to scale up to include the entire taxi fleet of over 21000 taxis in the country for improved accuracy. The LTA solution, however, has been severely suffering from the data sparsity issue all the time. A particular road segment may not have any taxis passing by at certain time slots, resulting in incomplete road coverage. According to our statistics on the Singapore taxi dataset (short for *SG dataset* and see more details in Section 6.1), in average less than 30 percent road segments (out of 58356 in total) are covered by taxi data in 15 mins time slot. The current LTA solution applies space interpolation to infer the traffic of uncovered roads, which delivers poor accuracy [9].

*Non-Linearity in Traffic Estimation.* Other than the simple interpolation method applied by the LTA, there have been many studies that exploit spatial correlation among road segments in order to complete the traffic estimation. The traffic of nearby road segments is mutually influenced, and thus their traffic conditions are highly correlated [23], [40], [52]. We term it as *traffic correlation* in this paper. Previous works explicitly or implicitly model the traffic correlations linearly mainly to reduce the tremendous computation overheads, and recover the complete traffic conditions by exploiting techniques like regression [41], [48], matrix factorization [39], tensor decomposition [33], [37], and compressive sensing [23], [52]. The practical traffic, however, is affected by a number of factors, e.g., intricate road network, traffic regulations, traffic lights, mixed traffic flows, weather condition, etc., and far more complex than linearly.

To better capture inscrutable traffic correlations, some recent works suggest building a non-linear model by ANN [4], [24], [35]. The ANN model is remarkable in representing complex patterns from imprecise data, and enables us to capture detailed traffic correlations for each road segment. Specifically, for road segment $r_i$ and its $n$ correlated road segments, we can build a three-layer feed-forward neural network to model their traffic correlations. The desired output of the model is traffic condition $v_i$ of $r_i$, while the input layer consists of traffic conditions $v_j$ of each correlated road segment $r_j$. The hidden layer of the model contains $(n+1)$

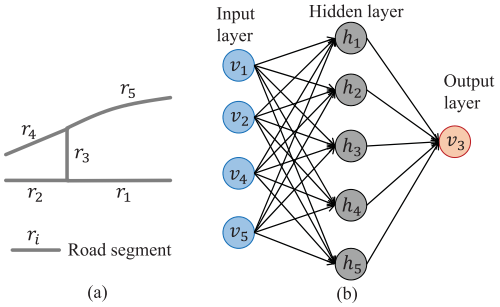



Fig. 1. (a) A simple road network. (b) The three-layer ANN model for capturing the traffic correlations between road segment $r_3$, $r_1$, $r_2$, $r_4$, and $r_5$.

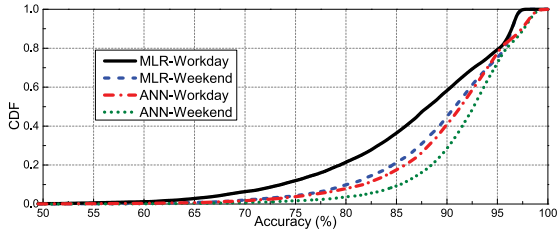



Fig. 2. CDF of MLR and ANN model accuracy on capturing the traffic correlations.

hidden unit[2] $h_k$. Both hidden and output units can use a standard hyperbolic tangent (i.e., `tanh`) activation function. Therefore, the ANN model includes $n \times (n+1)$ input-to-hidden parameters, $(n+1) \times 1$ hidden-to-output parameters. With sufficient training data of $m$ samples, a standard back-propagation algorithm can be adopted to train the ANN model by minimizing the mean square error as the optimization objective. The computation complexity to train an ANN model is $O(m * n^2)$.

For example, assuming the correlated road segments of road segment $r_3$ as depicted in Fig. 1a are $r_1$, $r_2$, $r_4$, and $r_5$, we can build the three-layer neural network as shown in Fig. 1b, which includes 4 input units, 5 hidden units, and 1 output unit. We train the model using historical traffic data, and can use the derived model to connect $r_3$'s traffic condition $v_3$ with the traffic conditions of $r_1$, $r_2$, $r_4$, and $r_5$.

To examine the advantage of non-linear models over the linear ones, we compare the traffic estimation accuracy using ANN model and using a representative linear model, i.e., multiple linear regression (MLR) [22], based on the *SG dataset*. The MLR model has been widely used for traffic correlation modeling in existing works [5], [23], [31], which represents traffic condition of one road segment as a linear combination of correlated road segments. Specifically, for a target road segment $r_i$ and its $n$ correlated road segments $r_j, j = 1, 2, \ldots, n$, we can build the MLR model to capture their traffic correlation to predict $r_i$'s traffic condition $c_{r_i}$ using the following equation:

$$c_{r_i} = \beta_0 + \sum_{j=1}^{n} \beta_j \times c_{r_j},$$

where $\beta_0$ and $\beta_j$ are model coefficients [22]. With sufficient training data, we can determine the MLR model using least-square method. Similar to the ANN model, the MLR model also takes the traffic conditions of $n$ correlated road segments as input to predict $r_i$'s traffic condition. For comparison, we randomly select 500 road segments for each road type from the Singapore road network (see details in Section 6.1). For each test road segment, we select its directly connected road segments for building both the ANN model and the MLR model to capture traffic correlations. The accuracy is derived from the estimations of ANN/MLR models and ground truth. The statistics in Fig. 2 show that ANN model has much higher accuracy than MLR model on both workdays and weekends. The gap is even wider on workdays, which experience much heavier and more complex traffic. In addition to high-level statistics, we compare their modeling capability on individual road segments. Specifically, for one road segment $r_i$, if ANN delivers higher accuracy than MLR we consider ANN beats MLR on $r_i$, and vice verse. We calculate the percentage of the road segments on which ANN wins and find that ANN outperforms MLR for 73.5 and 78.9 percent of all test road segments on workdays and weekends, respectively. In particular, their accuracy differences larger than 1 percent account for 69.5 and 76.9 percent for workdays and weekends, respectively. Moreover, we also analyze the other 26.5 and 21.2 percent cases where MLR performs better than ANN. In these cases, the accuracy differences greater than 1 percent are no more than 53 percent. These results show the gain of using ANN models to accurately capture the complex traffic correlations.

*Challenges.* While ANN based traffic correlation modeling provides clear improvement in estimation accuracy, it is challenging to introduce the ANN modeling for large scale estimation. First, fine ANN modeling for traffic correlations is non-trivial. A proper model needs to determine the correct correlated road segments, which involves various combinations in the road network space and thus results in huge computations. In addition, the random movements of taxis bring uncertain availability of traffic samplings [5], which requires dynamic ANN modeling during the estimation stage. The inherent complexity in building ANN models will inevitably introduce tremendous computation overheads, especially when we scale the traffic estimation to the entire city that involves tens of thousands of road segments and the related ANN modeling. Such computation overheads may overwhelmingly degrade the timeliness of traffic estimation.

## 3 Think Like a Graph

To address the challenges, we present Graph-parallel Processing based Traffic Estimation - GPTE. In this section, we detail how GPTE represents road network data as a property graph and enables non-linear correlation modeling based traffic estimation on the property graph.

### 3.1 Graph Representation

We model the underlying road network for traffic estimation as a property graph `G(V,E)`, where road segments are represented as vertices and edges are formed between any two physically connected road segments. Fig. 3 depicts the corresponding property graph for the simple road network presented in Fig. 1a. As the traffic conditions of connecting road segments are mutually influenced, the edges are thus bidirectional. In such a property graph, each vertex $r_i$ owns static properties (e.g., road type, road name, geographic

2. In this paper, we empirically set the number of hidden units in the hidden layer as $(n+1)$ to balance the computation overhead and prediction accuracy. We find that if we further increase the hidden unit number, e.g., $(n+2)$, the prediction accuracy improvement is less than 0.5 percent, while the ANN training time increases at least 1 second. However, even the 1-second time increase for each individual ANN model training could introduce about 81 minutes overall traffic estimation delay, which prohibits the near real-time traffic estimation.

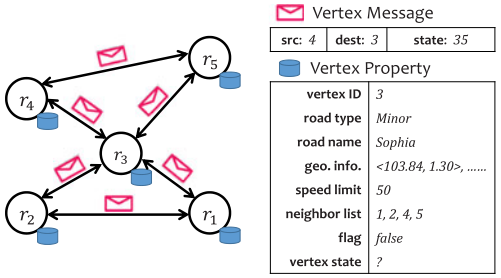



Fig. 3. Road network data as a property graph.

information, speed limit $v_i^{\max}$, neighbor list $\mathcal{N}_i^h$ within $h$ hops) and dynamic properties (e.g., `flag` and vertex state $v_i$). Specifically, geographic information contains a series of longitude and latitude coordinates to describe the locations of one road segment, with which we can match traffic samplings of taxis to appropriate road segments they traveled on. Speed limit $v_i^{\max}$ is the permissible maximum travel speed on road segment $r_i$, which can be acquired from the transport agency. Neighbor list $\mathcal{N}_i^h$ stores neighboring vertices of $r_i$ within $h$ hops. Vertex state $v_i$ is current traffic condition of $r_i$, and `flag` indicates whether $v_i$ is known or not. The users can easily incorporate other properties to the vertices if necessary. In graph-parallel processing frameworks, each vertex is able to send (and receive) messages along edges to (from) neighboring vertices. Each message contains source vertex ID, destination vertex ID, and source vertex state $v_i$. Based on message propagation and local vertex computation with messages, different correlation models and inference algorithms can be implemented on the property graph.

### 3.2 Dynamic Correlation Modeling

*Basic ANN Modeling*. Based on the property graph, a straightforward approach to embedding non-linear traffic correlations into traffic estimation is to pre-learn an ANN model for each vertex[3] and use this model for online vertex state inference. For vertex $r_i$, we can connect traffic condition $v_i$ with its immediate neighbors in $\mathcal{N}_i^1$ and build an ANN model to capture their traffic correlations for inferring $v_i$. Ideally these ANN models can be learned offline to lessen the computation burden of online traffic estimations. However, it cannot work in practice due to the dynamics of the property graph. As taxis randomly sample road traffic conditions, the vertices annotated by sufficient probe taxis change from time to time, resulting in time-evolving graph. Such dynamics cause uncertain availability of input vertex states in the ANN models and as a result the fixed ANN model may not work when the vertex states are missing. Fig. 4 presents a comparative example. Fig. 4a plots the traffic samplings collected at 9:00 AM in a 15 mins time slot, and Fig. 4b shows the corresponding property graph, where vertices annotated by $\geq 5$ taxis are considered credible and colored in red. Similarly, Fig. 4c shows the property graph at 10:00 AM, where we observe substantial differences of the annotated vertices when compared with Fig. 4b.

*Dynamic ANN Modeling*. To deal with dynamics in the time-evolving graph, we propose the *dynamic correlation modeling*. Instead of maintaining fixed ANN models, GPTE builds instant correlation models during online traffic estimation stage and performs traffic estimation via iterative message propagation and vertex state inference on the graph.

3. In the following, we use *vertex* and *road segment* interchangeably.

Within each time slot, GPTE associates the traffic samplings from taxis to vertices according to their locations. For a vertex $r_i$ visited by sufficient number, e.g., $\geq \lambda$, of taxis, GPTE annotates its state $v_i$ as the average of all travel speeds and sets `flag` as *true*. Otherwise, GPTE keeps $v_i$ as unannotated and sets `flag` as *false*. GPTE iteratively infers unknown vertex states from annotated vertex states via dynamic ANN modeling, which consists of message propagation, correlated vertex selection, and ANN based vertex state inference. These procedures are repeated for iterations until all vertex states are updated. The final output is the set of all vertex states, which corresponds to the traffic conditions of the entire road network. We detail each procedure in the following.

*(1) Message propagation*. In each iteration, only annotated vertices send messages to neighbors along edges, and all vertices receive messages. Each vertex $r_i$ will record the annotated neighbors in a set $\mathcal{C}_i$ based on the received messages. The unannotated vertices make use of these messages to infer their own states via dynamic correlation modeling. Specifically, for each unannotated vertex $r_i$, it will select several most correlated vertices from the annotated neighbors that have sent messages to $r_i$, and then build an instant ANN model to infer its own state $v_i$.

*(2) Correlated vertex selection*. Although we can simply treat all items in $\mathcal{C}_i$ as the correlated vertices of $r_i$ to build an ANN model, it may result in poor inference accuracy when considering uncorrelated vertices. To balance the computation overhead and model accuracy, GPTE only selects the most $\kappa$ correlated vertices from $\mathcal{C}_i$ for ANN modeling.

We select correlated vertices for $r_i$ from a data perspective. Specifically, we measure the traffic characteristics of each vertex using its historical traffic data, and determine $\kappa$ most correlated vertices for $r_i$ according to the correlations of their traffic data. In practice, one vertex may have no direct impact on the target vertex $r_i$'s traffic state $v_i$, while it may implicitly affect $v_i$ when combined with several other vertices. Identifying such a combination of vertices from the graph space is computationally complex. GPTE adopts a modified feature selection algorithm, mRMR [28], to efficiently approximate the optimal correlated vertex selection. mRMR is able to maximize the relevance while minimizing the redundancy among selected features.

We treat vertex $r_j \in \mathcal{C}_i$ as a *feature* for inferring the state $v_i$ of $r_i$, and use *mutual information* as the criteria to measure non-linear traffic correlations between vertices. For vertex $r_i$, we treat its state $v_i$ as a random variable $X_i$. Thus its entropy is defined as $H(X_i) = -\sum_{x \in X_i} P(x) \log(P(x))$, where $x$ denotes a specific value of $X_i$, and $P(x)$ denotes the probability of $x$ over all possible values of $X_i$, which can be calculated from historical traffic data. The mutual information between two vertices $r_i$ and $r_j$ is measured as

$$I(X_i; X_j) = \sum_{x \in X_i} \sum_{y \in X_j} P(x, y) \log \left( \frac{P(x, y)}{P(x)P(y)} \right),$$

where $P(x, y)$ is the joint probability of $x$ and $y$. $I(X_i; X_j)$ is used to measure the dependency between $X_i$ and $X_j$. The mRMR algorithm can be further updated by using the mutual information and the selection criterion in Equation (1) to evaluate each unselected feature [11], [28]:

$$NJ(X_k) = NI(X_k; X_i) - \frac{1}{|\mathcal{F}_i|} \sum_{X_j \in \mathcal{F}_i} NI(X_k; X_j). \quad (1)$$


(a) Traffic samplings at 9:00AM. (b) Graph snapshot at 9:00AM. (c) Graph snapshot at 10:00AM.

Fig. 4. Taxis report traffic samplings periodically along their routes, which result in time-evolving property graph: (a) Traffic samplings at 9:00AM of a typical workday, where green dots represent traffic samplings. (b) The corresponding property graph snapshot generated from (a). (c) The property graph snapshot at 10:00AM. For (b) and (c), red dots represent annotated vertices and black dots represent vertices of unknown states.

In particular, for a target road segment $r_i$ and the feature set $\mathcal{F}_i$ that has been selected, the first term in Equation (1) measures the mutual information of an unselected feature $r_k \in \mathcal{C}_i$ and target $r_i$, which is denoted as *dependence*, while the second term measures the average redundancy of $r_k$ and the already selected features in $\mathcal{F}_i$, which is denoted as *redundancy*. These two terms together determine whether $r_k$ should be further included in $\mathcal{F}_i$.

Different from the original mRMR algorithm in [28], we normalize the mutual information $I(X_k; X_i)$ by $H(X_i)$ according to [11] and define $NI(X_k; X_i) = \frac{I(X_k;X_i)}{H(X_i)}$, so that the normalized mutual information $NI(X_k; X_i) \in [0, 1]$, because the entropy of a feature itself could vary greatly. However, $H(X_i)$ cannot be used to normalize $I(X_k; X_i)$ directly, since $\frac{I(X_k;X_j)}{H(X_i)}$ may not fall in the range $[0, 1]$. Considering $0 \leq I(X; Y) \leq \min\{H(X), H(Y)\}$, we can further define $NI(X_k; X_j) = \frac{I(X_k;X_j)}{\min\{H(X_k),H(X_j)\}} \in [0, 1]$. Therefore, for a target road segment $r_i$, we iteratively select the feature $r_k \in \mathcal{C}_i$ that maximizes Equation (1), implying that $r_k$ has a large dependency with $r_i$ and small redundancy with current $\mathcal{F}_i$. We repeat this process until $\kappa$ features are selected or $NJ(X_k) \leq 0$.

*(3) ANN based vertex state inference.* Once the correlated vertices $\mathcal{F}_i$ are selected, the unannotated vertex $r_i$ can locally train an ANN model to capture its traffic correlations with the correlated vertices in $\mathcal{F}_i$ using their historical traffic speeds. Specifically, we normalize the historical traffic speeds by comparing with speed limit $v^{\max}$ of each vertex respectively during the ANN training. After successfully learning the correlation model, vertex $r_i$ feeds the normalized states of vertices in $\mathcal{F}_i$, extracted from received messages, into the ANN model to infer its own state $v_i$. Similarly, we normalize the vertex state $v_j$ of $r_j \in \mathcal{F}_i$ as $\frac{v_j}{v_j^{\max}}$ using its speed limit, and recover the state of vertex $r_i$ as the product of model inference and speed limit $v_i^{\max}$. The normalization can unify the speed scales of road segments in different road types and transform the input data to the range $[-1, 1]$, where the activation function `tanh` has the best non-linear transformation capability. Note that vertex $r_i$ only uses the built ANN model once in current time slot and it may need to learn a new model with different correlated vertices in next time slot due to the evolution of property graph. Finally, vertex $r_i$ updates its state as annotated and changes `flag` as *true*. In the subsequent iterations, $r_i$ will send messages to its neighbors for inferring other unknown vertex states.

## 4 Putting Things Into Spark

Ideally GPTE should implement the algorithms introduced in previous section into an integrated computing framework that supports both data-parallel and graph-parallel computation. The emerging cluster computing framework Spark [44], which builds on the abstraction *Resilient Distributed Datasets (RDDs)*, supports such requirements. In addition to dataflow operations, the graph processing engine GraphX [14] is built atop of Spark, which represents graph-structured data as a property graph including a pair of vertex RDDs and edge RDDs and embeds graph computations as specific *join-map-group-by* dataflow operators on these RDDs.

*Basic Implementation.* GPTE is implemented on the Spark and makes use of some GraphX APIs. Fig. 5 illustrates the execution flow of GPTE in a Spark cluster, which adopts the Hadoop distributed file system (HDFS) for distributed data storage. In the cluster, one machine is selected as the *Master* to coordinate parallel computing among other machines, i.e., *Workers*. GPTE first inputs the road network data from the HDFS to construct the property graph (via `Graph[V,E]()`). Without any optimization, the property graph is distributed to worker machines using the default graph partitioner of GraphX (via `partitionBy`), which exploits a hashing function to evenly distribute vertices and edges among all workers for load balance. GPTE then continuously conducts traffic estimation from each time slot of traffic samplings. It iteratively lets annotated vertices send messages to neighbors (via `sendMsg`) and then applies the received messages to the property graph (via `joinVertices`), which allows unannotated vertices to infer their own states (via `vprog`). The vertex program `vprog` embeds correlated vertex selection and ANN modeling based vertex state inference, as mentioned in Section 3.2. Finally the set of all vertex states is dumped to the HDFS for traffic visualization.

To improve the system efficiency, we propose (1) a geography-aware partitioner that optimizes the placement of edges/

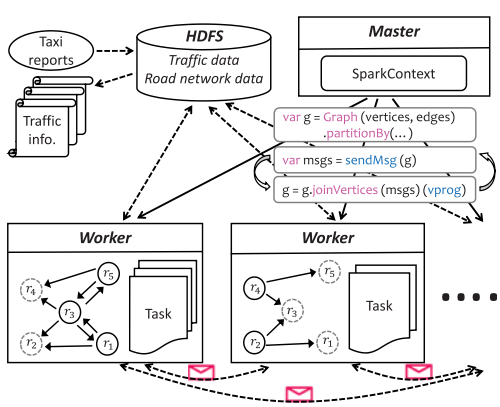
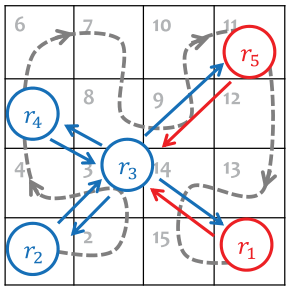



Fig. 5. The execution flow of GPTE in a Spark cluster, which consists of one master and some workers. The property graph is distributed to workers for parallel processing. The solid lines represent command flows and dashed lines represent data flow within the cluster.




(a)

```
def xy2key (n: Int, x: Int, y: Int) : Int {
  var rx, ry, key=0, s=n/2
  do {
    rx = x & s
    ry = y & s
    key += s * s * ((3 * rx) ^ ry)
    // rotate a quadrant appropriately
    if (ry == 0) {
      if (rx == 1) {
        x = n - 1 - x
        y = n - 1 - y
      }
      swap (x, y)   // swap x and y
    }
    s /= 2
  } while (s > 0)
  key
}
```

(b)

Fig. 6. (a) Illustration of Hilbert space-filling curve based graph partitioning, where the property graph in Fig. 3 is divided into two partitions (in blue and red colors). (b) The pseudocode of key assignment for a given vertex location coordinate $(x, y)$.

vertices and traffic data among machines to reduce unnecessary communications across different worker machines; and (2) an efficient information propagation scheme including multi-hop message broadcast and redundant message elimination to optimize the information exchange. These techniques can significantly reduce the intra-machine and inter-machine communications and finally improve the system efficiency.

### 4.1 Geography-Aware Graph Partitioning

Due to random edge/vertex assignments of hash-partitioner, the default graph partitioning schemes of GraphX may place neighboring vertices onto different machines, which incurs unnecessary communications and thus results in poor performance [19]. We thus expect a better graph partitioner tailored to the traffic estimation. Common graph partitioning approaches are categorized into *edge-cuts* and *vertex-cuts* [13]. The former divides vertices into disjoint clusters of nearly equal size while minimizing the number of edges spanning clusters, e.g., balanced edge-cuts in METIS [20]. Edge-cuts, however, results in many edge and vertex replicas that immensely consume memory. Vertex-cuts aims to divide edges into clusters with nearly equal size while reducing vertex replicas. Recent studies report that vertex-cuts is more effective than edge-cuts on processing the real-world graphs [13], [14]. Considering skewed power-law degree distribution of natural graphs, PowerGraph [13] proposes a vertex-cuts technique that heuristically places edges across machines to minimize vertex replicas. PowerLyra [8] proposes a hybrid-cuts that combines edge-cuts and vertex-cuts according to vertex degrees. Road network graphs, however, do not typically exhibit power-law degree distribution but a relatively flat distribution. These techniques thus are not directly applicable. To balance the advantages and disadvantages of above approaches, we choose vertex-cuts to avoid edge replicas. As the vertices geographically close to each other are more likely correlated in traffic and may exchange messages, we propose a geography-aware partitioner for edge/vertex placements to preserve spatial locality.

The partitioner used in GPTE first maps vertices to real-world geo-coordinates using the middle points of road segments. A simple and effective approach named Hilbert space-filling curve is used to index vertices. Space-filling curve allows one to map multi-dimensional data, e.g, 2-dimensional locations, to 1-dimensional keys that preserve spatial proximity [29]. Keys that are contiguous present nearby locations in space. We thus assign the space-filling curve keys to vertices based on their geographical information. The keys are then range-partitioned into disjoint clusters of nearly equal size. Edges are co-partitioned with vertices by assigning them the same keys as their source vertices. Fig. 6a illustrates how we divide the road network space into $4 \times 4$ cells using Hilbert space-filling curve and assign keys to vertices according to their location coordinates. Based on the keys, we divide the vertices and edges into two partitions. Fig. 6b shows the pseudocode of key assignment. There are some other techniques, e.g., Quad-tree or k-means, can be used to partition the graph while preserving the spatial proximity of vertices as well. It is, however, difficult for them to evenly control the partition sizes, and thus cannot balance the workloads among different machines (see the performance comparisons in Section 6.3), which will finally affect the system performance.

As expressways interact with other roads in traffic only at few entries and exits, we thus separately apply the spatial indexing technique to expressway vertices and the other vertices, and then unify the indexes of all vertices (and edges) according to their space-filling keys. We first re-index expressway vertices and then the others, both following the ascending order of their original space-filling keys. Finally, we obtain the continuous vertex keys and then range-partition vertices (and edges) given the desired number of partitions.

```
def sendMsg ( graph: Graph[V, E] ): RDD [Long, Map [Long, Double]] = {
  graph.vertices
       // only annotated vertices send messages
       .filter ( v.flag && v.nblist.length > 0 )
       // propagate messages to all neighbors
       .flatMap ( v.nblist.map ( ( nb, Map ( v.id -> v.state )))
       // aggregate messages targeting to the same vertex
       .reduceByKey ( _ ++ _ )
}
```

Fig. 7. Pseudocode of message propagation in Spark.

### 4.2 Efficient Information Propagation

We propose two techniques to optimize information propagation among vertices. One allows multi-hop message broadcast to speedup the state inferences of all vertices and the other eliminates redundant messages to further reduce communication costs. Fig. 7 shows the pseudocode that implements message propagation in the Spark.

*Multi-hop Message Broadcast.* Since road segments distant from each other may still be traffic correlated, it is necessary for one vertex to propagate its messages to vertices within multiple hops rather than only direct neighbors. To achieve this, a simple approach is based on the hop-by-hop message propagation, where each vertex receives a message and then forwards the message to the next hop until desired hops are reached. Such an approach, however, is not efficient. In GraphX, message propagation is implemented with `triplets` (a triplet contains an edge and its property, and the two vertex properties), which are derived through join operations on vertex RDDs and edge RDDs. As join operation has to be repeated for every hop, the hop-by-hop message propagation introduces excessive computations. We propose multi-hop message broadcast instead, which allows one vertex to broadcast messages to vertices within $h$ hops.

GPTE maintains a `broadcast variable`[4] $\mathcal{B}_{nb}$, which stores the 1-hop neighbors for each vertex. Each vertex $r_i$ in

4. A broadcast variable is a static lookup table and a copy is maintained in each machine to facilitate data access in the Spark.

different machine can easily access the variable $\mathcal{B}_{nb}$ and build its $h$-hop neighbor list $\mathcal{N}_i^h$. For vertex $r_i$, $\mathcal{N}_i^1$ includes its immediate neighbors, and the $h$-hop ($h > 1$) neighbor list $\mathcal{N}_i^h$ is recursively built as $\mathcal{N}_i^h = \bigcup_{r_j \in \mathcal{N}_i^{h-1}} \mathcal{N}_j^1$. Based on $\mathcal{N}_i^h$, vertex $r_i$ can directly send messages to its $h$-hop neighbors via `map` operator. To reduce the communication costs, GPTE first combines all messages targeting to the same vertex and then sends the aggregated message. Fig. 7 shows that GPTE realizes the message broadcast through the *map-reduce-by* operation in the Spark. GPTE applies the $h$-hop messages to the vertices with join operation only once, which completes message propagation with constant overhead instead of being proportional to the hop count.

*Eliminating Redundant Messages.* By default, annotated vertices propagate messages to their neighbors and at the same time all vertices receive the messages. As a matter of fact, such messages are only useful for unannotated vertices for inferring their states but redundant for the annotated vertices. Therefore, it is desired that the annotated vertices only send messages to unannotated vertices so that a large amount of unnecessary message exchanges can be saved.

We propose redundant message elimination for GPTE to avoid unnecessary computation and communication costs. The key idea is that each annotated vertex tracks the statuses of its $h$-hop neighbors and only sends messages to those unannotated neighbors in the subsequent iterations. Directly sending vertex status to other vertices in a distributed setting will introduce extra communication overhead, so GPTE implicitly infers vertex status by leveraging the fact that only annotated vertices propagate messages. If an annotated vertex $r_i$ receives a message from vertex $r_j$, it means that $r_j$'s state is already known and thus $r_i$'s messages to $r_j$ are redundant. Each vertex $r_i$ maintains an unannotated neighbor list $\overline{\mathcal{N}_i^h}$, which is initially copied from $\mathcal{N}_i^h$ in each time slot. For each annotated vertex $r_i$, it removes a neighbor $r_j$ from $\overline{\mathcal{N}_i^h}$ once it receives a message from $r_j$. Similarly, $r_i$ will also be removed from $\overline{\mathcal{N}_j^h}$ by vertex $r_j$ as $r_i$ sends messages to $r_j$ as well. In each iteration, $r_i$ only propagates messages to the neighbors in $\overline{\mathcal{N}_i^h}$. With more and more vertices annotated in later iterations, there are much fewer messages to be processed and thus the system efficiency is improved. To implement this idea in the Spark, GPTE replaces the neighbor list, i.e., `v.nblist` in Fig. 7, as $\overline{\mathcal{N}_i^h}$ and dynamically maintains the list for vertex $r_i$ during runtime.

## 5 Other Optimizations

As real-world traffic data are inherently sparse and noisy, GPTE adopts some data processing techniques to preprocess the traffic data before putting them to the estimation stage.

*Aggregate Travel Speeds to Alleviate Data Sparsity.* To derive an intact dataset for traffic correlation modeling, we compress traffic dataset according to time of the day and day of the week (i.e., Monday, $\cdots$, Sunday). We divide the entire day into a series of time slots with size of 15 mins. For each road segment $r_i$, we map each of its associated traffic samplings in the traffic dataset to a group according to time slot of the day and day of the week, and then average the reported travel speeds of the same group as the general traffic condition of $r_i$ for the specific time slot and day. We thus obtain a compact time series $\mathcal{P}_i$ containing $\frac{24\times 60}{15} \times 7 = 672$ traffic speeds. In case there still exist missing values, we use the average of previous 4 time slots in $\mathcal{P}_i$. We name $\mathcal{P}_i$ as the *speed profile* of road segment $r_i$ as it describes the general traffic speeds of $r_i$ in history. To avoid expensive data movements across machines, we save all speed profiles as a `broadcast variable` $\mathcal{B}_{sp}$ in the Spark.

*Smooth Traffic Speeds to Filter Out Noises.* The traffic samplings could be noisy due to inaccurate GPS reports of taxis, and thus the resulting speed profiles could be noisy as well. To filter out noises, we smooth the traffic speeds. For road segment $r_i$, we apply the exponential smoothing technique [32] to each day of the week for its speed profile, $\mathcal{D}_j \in \mathcal{P}_i, j = \{1, 2, \cdots, 7\}$, respectively. We apply the following equation to smooth the traffic speeds in $\mathcal{D}_j$:

$$s_{j,t} = \alpha_i \cdot x_{j,t} + (1 - \alpha_i) \cdot s_{j,t-1} \quad (t = 2, 3, \ldots, 96),$$

where $\alpha_i$ is the smoothing factor, $x_{j,t}$ is the raw traffic speed of the $t$-th time slot in $\mathcal{D}_j$, $s_{j,t}$ is the corresponding smoothed speed, and $s_{j,1} = x_{j,1}$. We determine the best $\alpha_i$ value for $r_i$ using the least square method by solving

$$\alpha_i = \arg\min_{0 < \alpha_i \leq 1} \sum_{j=1}^{7} \sum_{t=1}^{96} (s_{j,t} - x_{j,t})^2.$$

*Incorporate Traffic Patterns to Improve Performance.* As urban traffic is not consistent across time and days and exhibits various patterns [5], we thus incorporate such traffic patterns in the ANN modeling. Specifically, we mine traffic patterns for each road segment $r_i$ using its speed profile $\mathcal{P}_i$. We first split and compress $\mathcal{P}_i$ into two time series $\mathcal{P}_i^W$ and $\mathcal{P}_i^{NW}$ (both containing 96 time slots and $96 = 24 \times 60/15$) according to its time slot and the type of the day, which represent general traffic speeds of $r_i$ on workdays and weekends, respectively. We then separately apply the top-down time series segmentation algorithm [21] on $\mathcal{P}_i^W$ and $\mathcal{P}_i^{NW}$ to divide the 96 time slots into consecutive time slot groups, each of which represents a traffic pattern and preserves similar varying trend of traffic speeds. To guarantee that we have adequate traffic speeds in each traffic pattern for ANN training, we set the minimum size of a time slot group as $\omega$. The segmentation algorithm repetitively divides time slot series until each group cannot be separated any more. During online traffic estimation, GPTE builds the ANN model for each vertex only using traffic speeds belonging to the same traffic pattern, which avoids irrelevant training data that may lead to overfitting.

## 6 Evaluation

We conduct extensive trace-driven experiments with real-world traffic dataset to evaluate GPTE. We first describe the experimental setup, then evaluate the estimation accuracy of GPTE with comparisons to baseline methods, and finally run detailed experiments to evaluate the GPTE design.

### 6.1 Experimental Setup

We use the road network data and two months of taxi data provided by the LTA to evaluate GPTE in a cluster.

*Road Network.* The road network covers the whole main island of Singapore and contains all road segments. In the road network, two driving directions of a road are separately represented. LTA divides all roads into 58356 road segments for traffic analysis purpose. These road segments are categorized according to their types as expressway (=2404), major

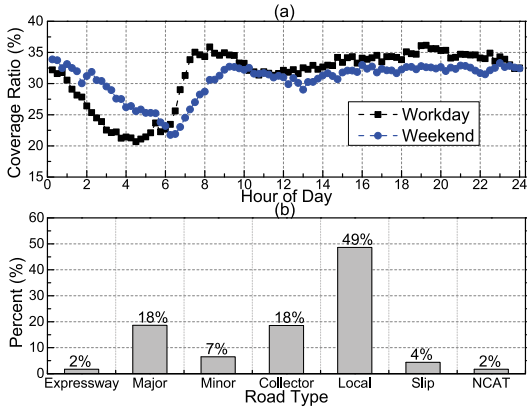



Fig. 8. (a) Road coverage by taxis across the time of a day. (b) Composition of uncovered road segments.

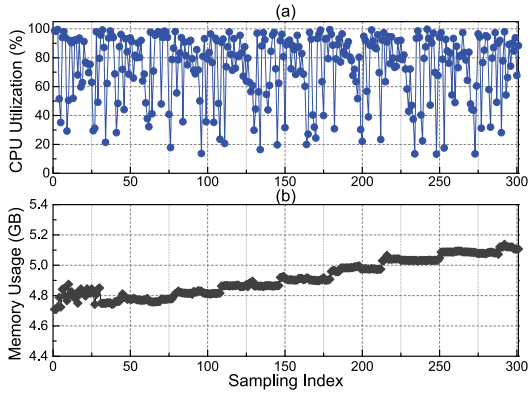



Fig. 9. The CPU utilization and memory usage of one worker machine.

(=14493), minor (=4814), collector (=11140), local (=21620), slip links (=3234), and NCAT (non-category, =651).

*SG Dataset*. The dataset contains traffic samplings collected from more than 12000 taxis covering the whole Singapore road network in July and August of 2015. Each taxi sends back a report every 30 seconds, which includes a timestamp, GPS location, travel speed, direction, status (*available* or *busy*), etc. About 10 million reports are collected each day.

To prepare a clean and correct dataset, we adopt an accurate map matching algorithm [27] to match each traffic sampling to the road segment a taxi actually traveled, which filters the errant traffic samplings due to GPS noises as well. In addition, we only keep the traffic samplings with taxi status as *busy* for experiments as they reflect more representative conditions of normal traffic. We derive the road coverage by the taxi data, and average the two month data into workday and weekend in Fig. 8a. We use a time slot of size 15 mins. For each time slot, the road coverage is measured as $\frac{\#\ of\ road\ segments\ with\ taxi\ samplings}{\#\ of\ all\ road\ segments}$. From Fig. 8a, we see the road coverage varies across the time of a day, and the average coverage ratios are only 32 and 29 percent for workday and weekend, respectively. Among the road segments without taxi samplings, we summarize its composition of road types. Fig. 8b gives the profile—most of them are local (49 percent), major (18 percent), and collector (18 percent) roads since the three road types account for large proportions among all kinds of road segments.

For evaluations, we keep the traffic data from the last week of August for testing and all the rest as historical traffic. During the testing phase, we randomly select $\rho$ percents of all taxis in each time slot as *probe taxis* and use their traffic samplings for traffic estimation. GPTE makes use of the vertex states annotated by $\lambda \geq 5$ probe taxis to infer unknown vertex states on the property graph. We perform traffic estimations for 5:00 AM to 21:00 PM everyday, which is the typical time period that contains more traffic in Singapore.

*Evaluation Metrics*. During each time slot, we calculate average traffic speed $v_i$ of road segment $r_i$ using all traffic samplings and consider the speed $v_i$ annotated by $\geq 5$ taxis as the *ground truth*. The estimation accuracy on $r_i$ is defined as $accuracy = (1 - \frac{|v_i - \hat{v}_i|}{v_i}) \times 100\%$, where $v_i$ is the ground truth and $\hat{v}_i$ is the estimation. We use the execution time of traffic estimation to measure the computation efficiency.

As Fig. 8a depicts, $\sim$ 60-80 percent of road segments are not covered by taxis in one time slot, so we want to specifically evaluate how our traffic estimation approach performs over such uncovered roads. Because we do not have any traffic samplings (so no ground truth) on those uncovered roads, direct evaluation is not possible. Instead, we construct a subset $\mathcal{S}$ of 100 road segments, composed of different road types (the composition follows the percentage profiled in Fig. 8b). We take away the taxi data on $\mathcal{S}$ from the original dataset and input to traffic estimation approach, so no traffic samplings on road segments of $\mathcal{S}$ are used during traffic estimation. On the other hand, we can build ground truth for those 100 road segments from the taxi data we take away, and use that for accuracy evaluation. The estimation accuracy on $\mathcal{S}$ suggests how the estimation approach performs on the uncovered roads (which account for $\sim$ 60%-80% road segments in each time slot). We separately report the accuracy results for subset $\mathcal{S}$ as well as those for all road segments in Singapore where we have $\geq 5$ taxi speed annotations.

*The Spark Configuration*. We implement GPTE on the Spark 2.1.0 [3]. For GPTE, we set $h = 5$ to enable each vertex broadcast its messages to neighbors within 5 hops. We set $\kappa = 3$ and thus each vertex selects at most 3 correlated vertices to build the ANN model. Accordingly, we set $\omega = 30$ to guarantee sufficient traffic speeds in each traffic pattern for ANN training. These settings are empirically determined in order to balance the estimation accuracy and computation overhead.

Our evaluation environment consists of 5 machines forming a cluster. Each machine has 24 Intel Xeon(R) 3.07 GHz CPU cores and 24 GB of memory. All machines share a disk of size 2TB. All the road network data and taxi data are stored in the HDFS for distributed data processing. To run GPTE in the Spark, we set one machine as the master machine and treat all the 5 machines as the worker machines, which will perform the ANN modeling and traffic estimations. We have measured the resource utilizations of GPTE on the Spark. Specifically, we sample the CPU utilization and memory usage for each machine with an interval of 1 second. Since the resource utilizations of all machines are quite similar, we only visualize the CPU utilization and memory usage of one ordinary worker machine for 5 minutes in Fig. 9 for a clear illustration (The master machine will have slightly higher resource utilizations). During the whole course of execution, the average CPU utilization is about 73 percent while the maximum value can even approach 100 percent. Meanwhile, the memory usage is stable around 5 GB in Fig. 9b and the maximum memory usage could be about 8 GB ($\sim$ 34 percent of all available memory) at the last. These measurements demonstrate that the ANN modeling based traffic estimation is computation extensive and the Spark heavily consumes CPU and memory resources due to the storage of RDDs for quick

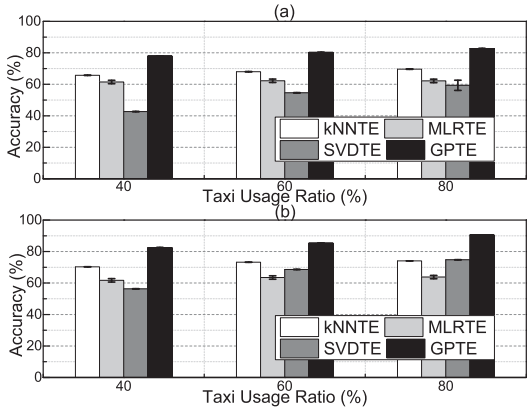



Fig. 10. Accuracy on (a) subset $\mathcal{S}$ and (b) those road segments with $\geq 5$ taxi speed annotations, with varied $\rho$.

computation. These results also imply that GPTE can well exploit the cluster resources and the power of Spark.

### 6.2 Performance Comparison

We compare GPTE with three baselines on the estimation accuracy and execution time. Although we cannot identify a proper way for direct comparison with Google Traffic due to the limit in accessing its original estimation results, we try to perform best effort analysis based on the available Google Traffic data and present at the end of this subsection.

*kNNTE*. This method enhances LTA solution [9] to achieve complete traffic estimations via kNN based interpolation, which measures traffic condition $v_i$ of road segment $r_i$ using average travel speed of all probe taxis passing by or average traffic speed of the top 10 nearest neighboring road segments (whose current traffic conditions are available) of the same road type as $r_i$ [6], [35] when no probes travel on $r_i$.

*MLRTE*. Some works explicitly capture the traffic correlations using MLR models. The models are learned from historical data, and later used for the online traffic estimation [5], [23], [31]. We implement the state-of-the-art MLR based method in [23] for performance comparison.

*SVDTE*. There exist some works implicitly exploiting traffic correlations in a traffic matrix to recover the missing traffic conditions [33], [37], [52]. Specifically, they first construct a traffic matrix consisting of traffic conditions of $m$ road segments in $t$ time slots and then exploit singular value decomposition (SVD) technique to recover missing values in the $m \times t$ matrix. We implement a typical SVD based method in [52] for comparison, which sets $t = 32$ for the best results.

We implement kNNTE in the Spark and run it in the cluster. As there are no cluster computing version of MLRTE and SVDTE, we implement and run them using multi-threading in a powerful HP Z440 Workstation that has 12 3.5 GHz Intel Xeon CPU cores and 32 GB memory. We try our best to optimize these methods to derive their best performances. For comparison, we explore how these methods perform with different amount of traffic data (i.e., varied taxi usage ratio $\rho$ as 40, 60, and 80 percent) and report estimation accuracies on subset $\mathcal{S}$ and on those road segments with $\geq 5$ taxi speed annotations in Fig. 10. All methods have lower accuracy on subset $\mathcal{S}$. This is because there is no taxi input from those roads contained in $\mathcal{S}$, and thus the traffic estimation of $\mathcal{S}$ is merely inferred from traffic samplings on other road segments. The accuracy on $\mathcal{S}$ reflects the traffic estimation performance on those $\sim$ 60%-80% uncovered roads in each time slot. We can see our method outperforms the three baselines on subset $\mathcal{S}$, and achieves high accuracy above 80 percent while the others only have accuracy around 60 percent when $\rho \geq 60\%$. Even when $\rho = 40\%$ with much fewer taxis for traffic monitoring, our method still achieves reasonably high accuracy about 78 percent while other methods only have accuracy $\leq 62\%$. It implies that GPTE can still work well even in a scenario like terrible weather conditions with a reduced number of available taxis. As a matter of fact, we do observe some time slots with much fewer taxis than usual in the dataset. On those road segments with $\geq 5$ taxi speed annotations, the accuracies of all methods increase, e.g., 73, 64, 74, and 90 percent for kNNTE, MLRTE, SVDTE, and GPTE, respectively, when $\rho = 80\%$. Relying on simple interpolation without considering traffic correlations among roads, kNNTE (i.e., the enhanced LTA solution) cannot get accurate estimations. Although MLRTE and SVDTE performs well and consider the traffic correlations, the linear modeling cannot promise good performances neither, with both accuracies lower than 75 percent. Benefited from non-linear traffic correlation modeling, GPTE outperforms the baselines on accuracy by 12 $\sim$ 34 percent as indicated by Fig. 10.

TABLE 1
Performance Comparison on Execution Time (in Seconds) with Different Taxi Usage Ratio $\rho$

| **Ratio** $\rho$ | kNNTE | MLRTE | SVDTE | GPTE |
|---|---|---|---|---|
| 40% | 5.3 | 25.5 | 60.8 | 41.0 |
| 60% | 4.4 | 35.7 | 61.6 | 37.7 |
| 80% | 3.8 | 48.0 | 62.1 | 33.4 |

We compare the execution times in Table 1. kNNTE runs the fastest due to its simplicity, while MLRTE and GPTE can accomplish traffic estimation within 1 minute. With more traffic data, the execution time of MLRTE and SVDTE increases while the other two run faster. This is because more traffic data lead to more initially annotated vertices, and the computation tasks for inferring unknown vertex states are thus reduced for kNNTE and GPTE. Even conducting ANN modeling with more taxi data when $\rho = 80\%$, GPTE still runs much faster than MLRTE and SVDTE, improving up to 46 percent. SVDTE accumulates sufficient traffic data for $t = 32$ time slots to perform SVD based missing traffic condition recovery. As a result, it introduces great estimation delay (i.e., $15 \times 32 = 480$ minutes) and cannot provide timely traffic information. The other methods can immediately estimate traffic conditions with instant traffic data.

Google Traffic provides the live traffic visualization, but it only gives 4 coarse traffic levels (from *Fast* to *Slow*) instead of providing detailed road traffic estimation to the end users. Besides, there is no interface[5] for public access of Google Traffic's historical or live traffic estimation data. As a result, it is impossible for us to directly compare it with GPTE on the traffic estimation accuracy. We try to manually measure the coverage of Google Traffic based on a selected region of size 7km $\times$ 5km in the downtown area of Singapore (which covers 713 road segments). We count the number of road segments with live traffic data from Google Traffic for four 15 mins time slots (during 17:30 PM - 18:30 PM) a day for one week, and calculate the coverage ratio. According to our statistics, while all

5. Although Google Maps Directions APIs enable the users to query the travel routes and travel durations, these APIs do not provide the detailed traffic estimation of any specific road segment.

(a) The whole road network. (b) Downtown area.

Fig. 11. Traffic estimation at peak hour 8:00AM of a typical workday: (a) The whole road network. (b) Downtown area where green, yellow, and red represent the traffic speed in *Normal*, *Slow*, and *Congestion*, respectively.

the 145 expressway and major road segments are covered, other roads have poor coverage, with an average coverage ratio about 37.8 percent. The overall estimation coverage of this busy downtown district is only 50.4 percent.

### 6.3 Detailed Evaluation

We present the traffic visualization and then evaluate GPTE and its design components. We fix the taxi usage ratio $\rho$ as 70 percent and report the estimation accuracy on all road segments with $\geq 5$ taxi speed annotations in this subsection.

*Traffic Visualization.* We visualize our traffic estimation results to LTA through a web service. For better understanding the general traffic conditions, we translate specific traffic speed $v$ (in *km/h*) into one traffic congestion level according to LTA's rules, i.e., *Congestion* ($v < 40$ for expressways, and $v < 20$ for others), *Slow* ($40 \leq v < 60$ for expressways, and $20 \leq v < 40$ for others), and *Normal* ($v \geq 60$ for expressways, and $v \geq 40$ for others). Fig. 11a depicts the traffic estimation snapshot at peak hour 8:00 AM of a typical workday. Even with a small number of probe taxis ($\sim$ 2500 in each time slot), our method can still derive the traffic conditions of the whole road network. Fig. 11b shows the traffic conditions for all roads in the downtown area, most of which are in poor traffic conditions due to commuter traffic.

*Overall Performance.* According to our statistics, the number of all taxis (probe taxis) ranges from 1388 to 4942 (972 to 3460) in each time slot throughout the testing week. On average, 17716 road segments ($\sim$ 30.4 percent of all) are traveled by taxis in a time slot. Fig. 12a presents the average estimation accuracy across the time of a day. GPTE achieves relatively higher and more stable accuracy on weekends than workdays. This is possibly because the traffic on weekends is more regular than on workdays and thus the traffic correlations are better modeled. The average accuracies for workdays and weekends are 87 and 88 percent, respectively. We observe obvious accuracy decrease in the time range [6:30 AM - 9:00 AM] and [17:30 PM - 19:00 PM] on workdays, which are the commuter rush hours in Singapore. Surprisingly, GPTE still achieves high accuracy above 83 percent in these rush hours that always contain heavy and complex traffic.

We plot the statistics of execution time of traffic estimations in Fig. 12b. The 90-percentile and 50-percentile execution time are 37 seconds and 33 seconds for workdays, and 38 seconds and 35 seconds for weekends, respectively. GPTE runs a bit longer on weekends as there are less taxi data and thus more state inference tasks. The overall average is 34 seconds, which means that GPTE can return traffic estimation results with a small delay. It is worthy to note that training an ANN model in the HP Z440 Workstation takes about 1.48 seconds, and it will need much more time to build ANN models for all road segments due to the large computation overhead, e.g., tens of minutes even we adopt the multi-threading technique, which is much larger than the time budget of 15 mins. Therefore, Fig. 12b demonstrates that GPTE can significantly improve the computation efficiency with high parallelism.

*Accuracy Across Different Road Types.* We report the accuracy performance of GPTE on different road types in Fig. 13. GPTE gets the best estimations for expressways with accuracy higher than 92 percent, and good estimation for major and slip roads (which bridge expressways and major roads) with accuracy $\sim$ 85 percent. The results for collector, local and NCAT roads are with slightly low accuracy ($\sim$ 80 percent) and large variances. The performance gap among different road types is caused by following reasons. On one hand, high-level roads, e.g., expressways, are usually

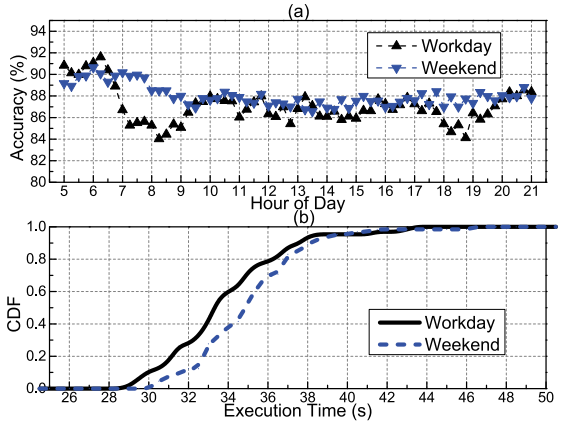



Fig. 12. Overall performance on (a) estimation accuracy across the time of a day and (b) execution time.

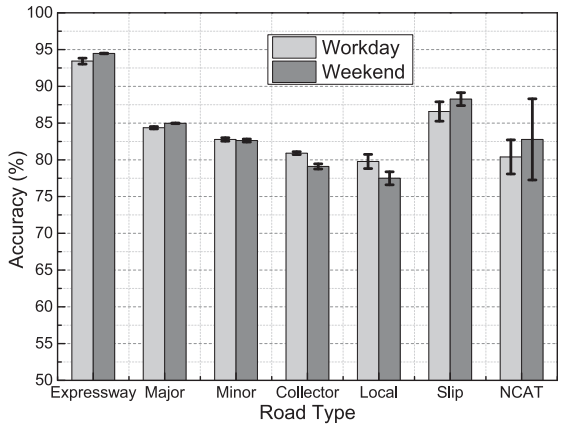



Fig. 13. Estimation accuracies and variances for different road types.

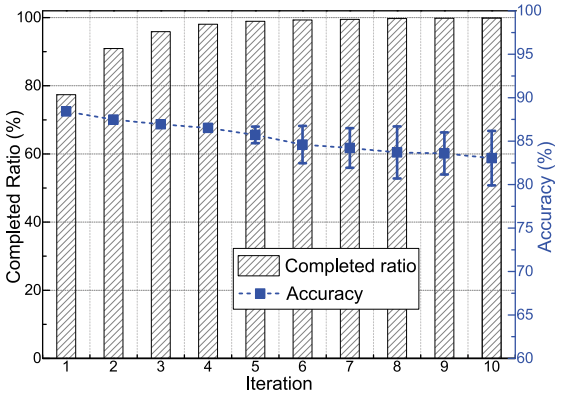



Fig. 14. The completed ratio and accuracy of traffic estimation in different iterations.

traveled by more vehicles and we can have sufficient traffic data to accurately capture their traffic correlations while we only have limited data on the low-level roads, e.g., local and collector roads are largely uncovered by taxis as shown in Fig. 8b. On the other hand, high-level roads tend to have fewer external influencing factors while vehicles are governed by many factors on the low-level roads, e.g., traffic lights exist on low-level roads but not on expressways.

*Accuracy Across Different Iterations.* GPTE makes use of message propagations for iterative vertex state inferences and we thus report the completed inference ratio (calculated as $\frac{\#\ of\ vertices\ with\ already\ known\ states}{\#\ of\ all\ vertices}$) and accuracy for each iteration. From Fig. 14, we see that the states of $> 90\%$ unannotated vertices can be inferred within 2 iterations and we can derive almost all ($\sim$99.9 percent) vertex states within 10 iterations. In the dynamic ANN modeling, estimation errors may be accumulated along with the message propagations. Fig. 14 shows that the traffic estimation indeed becomes less accurate and reliable (with a larger variance) as the iteration increases. However, benefited from the good generalization of the ANN model, the performance loss is limited, e.g., the average accuracy decreases by around 4 percent merely. In the future, we will study how to introduce a discounting factor for the inferred states that are further used for ANN predictions to reduce the influence of estimation error accumulation along with the message propagations.

Next we will show how the proposed optimization techniques help GPTE achieve timely and accurate traffic estimation.

*Benefit of Geography-Aware Graph Partitioning.* To understand the impact of data placement on system efficiency, we compare our geography-aware graph partitioner using space-filling curve technique with the Quad-tree based graph partitioner and three default graph partitioning schemes of GraphX, i.e., *RandomVertexCut* (Random), *EdgePartition1D* (EP1D), and *EdgePartition2D* (EP2D). The Quad-tree based method recursively subdivides a graph into four quadrants until the number of vertices in each quadrant is smaller than a threshold. This method preserves spatial proximity of vertices as well but it unevenly distributes vertices among partitions. The default graph partitioner assigns edges/vertices to different partitions by hashing source (or destination) vertex ID, resulting in random placements of edges/vertices among machines. In contrast, our scheme exploits vertex locations to place nearby edges/vertices to the same machine.

In order to compare the communication efficiency of different schemes, we count the number of cross-partition

TABLE 2
The Cross-Partition Messages for Different Graph Partitioning Schemes

| Scheme | Space-filling | Quad-tree | Random | EP1D | EP2D |
|---|---|---|---|---|---|
| Average ($\times 10^3$) | 11.8 | 31.3 | 145.1 | 144.9 | 145.2 |

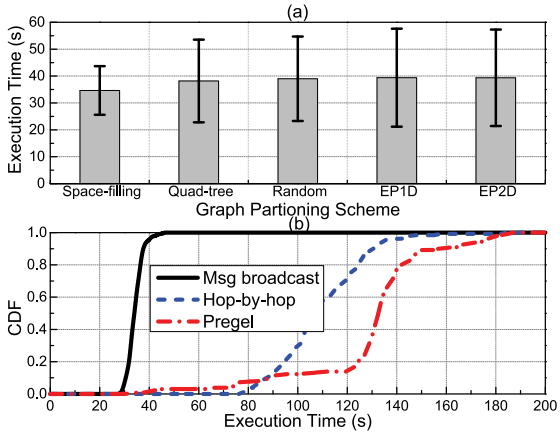



Fig. 15. Evaluation of (a) graph partitioning and (b) message broadcast.

messages (communication overhead). During each traffic estimation, about $153.3\,k$ messages in total are generated for information propagation among vertices, while there are on average $11.8\,k$, $31.3\,k$, $145.1\,k$, $144.9\,k$, $145.2\,k$ cross-partition messages for the five schemes respectively, as shown in Table 2. From the statistic, we see that space-filling method significantly outperforms other methods, with reduction on cross-partition messages by $4.4\times$ than Quad-tree and $11.4\times$ than the three default schemes of GraphX.

In addition to the comparisons on cross-partition messages, we use execution time as another metric to compare these five schemes since cross-partition messages could be costly and will explicitly affect the execution time. The comparison results are depicted in Fig. 15a. Compared to default schemes of GraphX, the benefit of exploiting vertex locations is clear that the proposed scheme reduces the execution time up to 6 seconds. When vertices that are geographically close to each other are assigned to the same partition, the data exchanges among different partitions are reduced. Although Quad-tree based scheme has also exploited geographical information of vertices, it cannot evenly assign vertices to partitions that lead to unbalanced loads among machines. Relying on the Hilbert space-filling curve technique, our scheme indexes vertices with consecutive keys and can evenly partition vertices among machines, which is more flexible and efficient. Besides, our scheme makes graph processing more stable when compared with other schemes on the variance of execution times.

*Benefit of Efficient Information Propagation.* Rather than propagating messages among vertices in a hop-by-hop manner, we propose the multi-hop message broadcast to accelerate information propagation. Indeed we can first learn $h$-hop neighbors for each vertex and then build another graph where vertices are still the road segments and edges are formed between each vertex and its $h$-hop neighbors. In this graph, each vertex can directly send messages to its original $h$-hop neighbors through the `triplets` in GraphX. We implement this idea using `Pregel` [25] API of GraphX. We compare the three message propagation

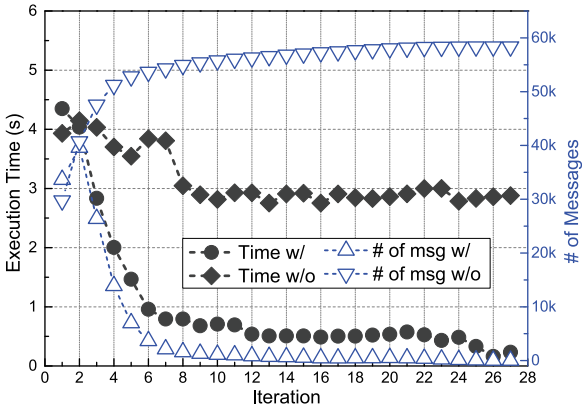



Fig. 16. The execution details of one sample traffic estimation.

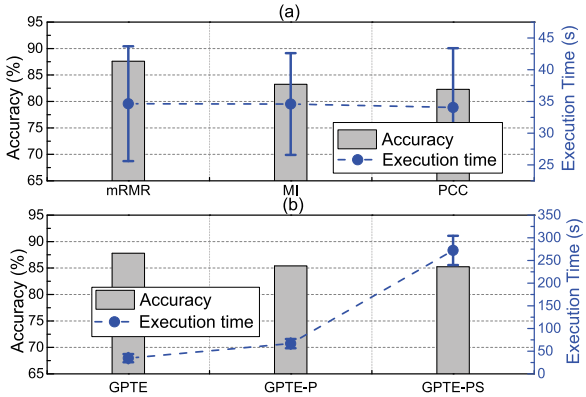



Fig. 17. Evaluation of various data processing techniques.

methods and plot the results in Fig. 15b. The average execution times of multi-hop message broadcast, hop-by-hop method, and Pregel based method are 34 seconds, 110 seconds, and 130 seconds, respectively. Our method significantly outperforms the hop-by-hop method and Pregel based method with gains of $3.2\times$ and $3.8\times$, respectively. The multi-hop message broadcast avoids repetitive join-operations between vertex RDDs and edge RDDs when compared to the hop-by-hop method, and meanwhile keeps the original road network structure rather than generating a particular graph as Pregel based method dose.

In addition, we propose redundant message elimination to further improve the efficiency of information propagation. By avoiding redundant messages to annotated vertices, GPTE can reduce communication costs. Fig. 16 shows the execution details of one sample traffic estimation. This job lasts for 27 iterations and we report the execution time and number of messages in each iteration for the scenarios *with* and *without* redundant message elimination. Without such a mechanism, there are more and more message exchanges as more vertex states are inferred along with the time. The execution time per iteration maintains for about 3 seconds. In contrast, eliminating redundant message significantly reduces the number of messages for the later iterations and execution time of each iteration is continuously decreased as shown in Fig. 16. According to our statistics, redundant message elimination reduces the average execution time from 101 seconds to 34 seconds, providing gains of $\sim 3\times$.

*Benefit of Data Processing Techniques.* We use a set of data processing techniques to optimize GPTE. We adopt a modified feature selection algorithm, mRMR, to select the $\kappa$ most correlated vertices during ANN modeling. For feature selection, metrics like *mutual information* (MI) and *Pearson correlation coefficient* (PCC) have been widely used [15]. To evaluate the effectiveness of our modified mRMR algorithm, we run GPTE with different feature selection methods. For correlated vertex selection using MI or PCC, we choose features with the $\kappa$ largest values. We show the comparisons in Fig. 17a. mRMR outperforms both MI and PCC by $\sim$4 percent on accuracy. The gain arises because mRMR selects correlated vertices as a whole rather than only considering individual correlation of each single vertex like MI and PCC do. To achieve that, mRMR introduces negligible computation costs.

In addition, we propose *data compact* to alleviate data sparsity, *data smoothing* to filter noises, and *traffic pattern mining* to avoid irrelevant training data. To evaluate performance contributions of these techniques, we gradually exclude these data processing techniques and run experiments with various GPTE versions, i.e., GPTE excluding traffic pattern mining (GPTE-P) and GPTE excluding both traffic pattern mining and data smoothing (GPTE-PS). As depicted in Fig. 17b, without such techniques the system runs much slower and gets worse accuracy. The data smoothing can reduce execution time from 272 seconds to 67 seconds with a gain of $4\times$, and traffic pattern mining can further improve the performance by up to $8\times$, i.e., execution time from 272 seconds to 34 seconds. Data smoothing reduces the variances of training data while traffic patterns ensure that the ANN training only uses relevant speed data. As a result, these data processing techniques not only speed up the ANN modeling but also together improve the estimation accuracy from 85 to 88 percent.

## 7 Related Work

*Traffic Monitoring and Estimation.* Previous works heavily rely on intrusive sensing infrastructures to monitor real-time traffic flows [24], traffic volumes [5], and traffic queues [30]. Due to the excessive deploying and maintenance costs, however, it is prohibitive to widely adopt them at city scale, which largely limits the coverage. Recent studies leverage instant location reports collected from probe vehicles as an efficient alternative for passive traffic monitoring. They utilize these reports to generate traffic map [17], [49], update digital map [36], predict vehicle travel time [6], [34], [50], predict taxi fare for a trip [6], monitor road surface conditions [10], [43], and model human mobility [42], [45]. These works, however, are often limited by insufficient probe vehicles due to privacy concerns or insufficient incentives, and thus suffer from limited monitoring coverage as well.

By exploiting traffic correlations among different road segments, some attempts were made to recover the missing traffic conditions through regression [41], [48], matrix factorization [39], multichannel singular spectrum analysis [51], tensor decomposition [33], [37], and compressive sensing [23], [52]. Subject to the tremendous computations for large scale traffic estimation, these methods explicitly or implicitly model the traffic correlations linearly and thus cannot derive accurate results. The practical traffic is influenced by many factors that make traffic correlations far more complex than linearly. In addition, some recent works [26], [46], [47] exploit multi-source data, e.g., taxi trajectory, loop detector data, truck data, and etc, to derive the traffic

conditions. Specifically, they build one model for each individual data source and integrate these models for multi-source data to obtain the final traffic estimation results, where each traffic model essentially assumes the traffic correlation to be linear. In this paper, we propose a non-linear model to accurately capture practical traffic correlations from taxi data and enable advanced traffic estimation at city-scale based on the graph-parallel processing design.

There exist a few works that adopt advanced models for traffic estimation and prediction [35]. They primarily rely on modeling the temporal correlation of traffic for each individual road segment using support vector machine [4], artificial neural network [24], and hidden Markov model [40]. These works, however, are mainly designed for specific roads, e.g., expressways, and implicitly assume sufficient amount of data can be collected. Although the emerging deep learning technique can be used for traffic estimation [24], it introduces a large number of extra parameters to be estimated and thus requires much more training data, which will greatly increase the computation complexity. The enormous computations involved in non-linear models strictly limit their applicability to large road networks. Our work differs from these works by considering the data sparsity issue and improving computing parallelism for city scale traffic estimation.

*Graph Analysis and Processing.* Various analysis can be conducted on the road network graphs, e.g, travel route planing [12] and spatial query [7]. These works involve only small dataset with negligible computation burdens. In contrast, non-linear correlation modeling based large scale traffic estimation requires much more computation efforts. The popular cluster computing framework Spark [44] serves as a powerful engine to handle continuously increasing graph scale and dataset, which could be adopted for satisfying application requirements on timeliness and efficiency [18]. Meanwhile, emerging graph-parallel processing frameworks, e.g., Pregel [25], GraphLab [13], and GraphX [14], show their capabilities in settling large scale graph analysis problems, e.g., cellular network analysis [19]. To the best of our knowledge, this is the first work that adapts traffic data analytics to the graph-parallel processing context. Our work implements and optimizes the advanced traffic estimation for city scale road networks in the latest cluster computing framework.

## 8 Conclusion

This paper presents GPTE for accurate and timely traffic estimation at city scale. GPTE proposes non-linear traffic correlation modeling and adapts the advanced traffic estimation to the graph-parallel processing paradigm through dynamic ANN modeling and efficient information propagation. A number of optimization techniques are proposed to improve the design. Experiment results from real-world traffic data demonstrate that GPTE significantly outperforms baseline solutions and can derive traffic estimation for the entire city in 34 seconds with accuracy as high as 88 percent.

As the future work, we could analyze mobility patterns of taxis and augment taxi probe based traffic sensing with crowdsourcing. Specifically, we could well design the routes for empty taxis that will migrate some taxis from popular road segments to those less popular ones, so as to improve the overall data coverage of the whole road network. With more initial vertex states, we can improve the traffic estimation accuracy further.

## Acknowledgments

This work is supported by a China NSFC grant (No. 61802261), a NSF grant from Shenzhen University (No.2018061), an ECS grant from the Research Grants Council of Hong Kong (No. CityU 21203516), and a GRF grant from Research Grants Council of Hong Kong (No. CityU 11217817), Singapore MOE Tier 2 grant MOE2016-T2-2-023, and NTU CoE grant M4081879.

## References

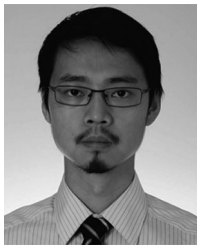
[1] LTA of Singapore. [Online]. Available: https://www.lta.gov.sg/, Accessed on: Aug. 2018.
[2] TrafficScan portal. [Online] https://www.onemotoring.com.sg/content/onemotoring/en/imap.html, Accessed on: Aug. 2018.
[3] Apache Spark. [Online]. Available: http://spark.apache.org/, Accessed on: Aug. 2018.
[4] M. T. Asif, J. Dauwels, C. Y. Goh, A. Oran, E. Fathi, M. Xu, M. M. Dhanya, N. Mitrovic, and P. Jaillet, "Spatiotemporal patterns in large-scale traffic speed prediction," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 2, pp. 794–804, Apr. 2014.
[5] J. Aslam, S. Lim, X. Pan, and D. Rus, "City-scale traffic estimation from a roving sensor network," in *Proc. 10th ACM Conf. Embedded Netw. Sensor Syst.*, 2012, pp. 141–154.
[6] R. K. Balan, K. X. Nguyen, and L. Jiang, "Real-time trip information service for a large taxi fleet," in *Proc. 9th Int. Conf. Mobile Syst. Appl. Serv.*, 2011, pp. 99–112.
[7] Z. Bolong, Z. Kai, X. Xiaokui, S. Han, Y. Hongzhi, and X. Zhou, "Keyword-aware continuous kNN query on road networks," in *Proc. IEEE 32nd Int. Conf. Data Eng.*, 2016, pp. 871–882.
[8] R. Chen, J. Shi, Y. Chen, and H. Chen, "Powerlyra: Differentiated graph computation and partitioning on skewed graphs," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, Art. no. 1.
[9] K. K. CHIN and C. W. LEE, "TrafficScan - bringing real-time travel information to motorists," *The LTA Academy (LTAA).*, vol. 1, no. 5, pp. 7–14, 2009.
[10] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, "The pothole patrol: Using a mobile sensor network for road surface monitoring," in *Proc. 6th Int. Conf. Mobile Syst. Appl. Serv.*, 2008, pp. 29–39.
[11] P. A. Estévez, M. Tesmer, C. A. Perez, and J. M. Zurada, "Normalized mutual information feature selection," *IEEE Trans. Neural Netw.*, vol. 20, no. 2, pp. 189–201, Feb. 2009.
[12] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag, "Adaptive fastest path computation on a road network: A traffic mining approach," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 794–805.
[13] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed graph-parallel computation on natural graphs," in *Proc. 10th USENIX Conf. Operating Syst. Des. Implementation*, 2012, pp. 17–30.
[14] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph processing in a distributed dataflow framework," in *Proc. 11th USENIX Conf. Operating Syst. Des. Implementation*, 2014, pp. 599–613.
[15] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn Res.*, vol. 3, no. Mar, pp. 1157–1182, 2003.
[16] R. Herring, A. Hofleitner, P. Abbeel, and A. Bayen, "Estimating arterial traffic conditions using sparse probe data," in *Proc. 13th Int. IEEE Conf. Intell. Transp. Syst.*, 2010, pp. 929–936.
[17] B. Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J.-C. Herrera, A. M. Bayen, M. Annavaram, and Q. Jacobson, "Virtual trip lines for distributed privacy-preserving traffic monitoring," in *Proc. 6th Int. Conf. Mobile Syst. Appl. Serv.*, 2008, pp. 15–28.
[18] T. Hunter, T. Moldovan, M. Zaharia, S. Merzgui, J. Ma, M. J. Franklin, P. Abbeel, and A. M. Bayen, "Scaling the mobile millennium system in the cloud," in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011, Art. no. 28.
[19] A. Iyer, L. E. Li, and I. Stoica, "CellIQ: Real-time cellular network analytics at scale," in *Proc. USENIX NSDI*, 2015, pp. 309–322.
[20] METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering. [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview, Accessed on: Aug. 2018.

[21] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *Proc. IEEE Int. Conf. Data Mining*, 2001, pp. 289–296.
[22] M. H. Kutner, C. Nachtsheim, and J. Neter, *Applied Linear Regression Models*. New York, NY, USA: McGraw-Hill/Irwin, 2004.
[23] Z. Liu, Z. Li, M. Li, W. Xing, and D. Lu, "Mining road network correlation for traffic estimation via compressive sensing," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 7, pp. 1880–1893, Jul. 2016.
[24] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, "Traffic flow prediction with big data: A deep learning approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 865–873, Apr. 2015.
[25] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 135–146.
[26] C. Meng, X. Yi, L. Su, J. Gao, and Y. Zheng, "City-wide traffic volume inference with loop detector data and taxi trajectories," in *Proc. ACM SIGSPATIAL*, 2017, pp. 1–10.
[27] P. Newson and J. Krumm, "Hidden Markov map matching through noise and sparseness," in *Proc. 17th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2009, pp. 336–343.
[28] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, pp. 1226–1238, Aug. 2005.
[29] H. Sagan, "Hilbert's space-filling curve," in *Space-Filling Curves*. New York, NY, USA: Springer, 1994, pp. 9–30.
[30] R. Sen, A. Maurya, B. Raman, R. Mehta, R. Kalyanaraman, N. Vankadhara, S. Roy, and P. Sharma, "Kyun queue: A sensor network system to monitor road traffic queues," in *Proc. 10th ACM Conf. Embedded Netw. Sensor Syst.*, 2012, pp. 127–140.
[31] Z. Shan, D. Zhao, and Y. Xia, "Urban road traffic speed estimation for missing probe vehicle data based on multiple linear regression model," in *Proc. 16th Int. IEEE Conf. Intell. Transp. Syst.*, 2013, pp. 118–123.
[32] R. H. Shumway and D. S. Stoffer, "An approach to time series smoothing and forecasting using the EM algorithm," in *J. Time Series Anal.*, vol. 3, no. 4, pp. 253–264, 1982.
[33] H. Tan, G. Feng, J. Feng, W. Wang, Y.-J. Zhang, and F. Li, "A tensor-based method for missing traffic data completion," *Transp. Res. Part C: Emerging Technol.*, vol. 28, pp. 15–27, 2013.
[34] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, "VTrack: Accurate, energy-aware road traffic delay estimation using mobile phones," in *Proc. 7th ACM Conf. Embedded Netw. Sensor Syst.*, 2009, pp. 85–98.
[35] E. I. Vlahogianni, M. G. Karlaftis, and J. C. Golias, "Short-term traffic forecasting: Where we are and where we're going," *Transp. Res. Part C: Emerging Technol.*, vol. 43, pp. 3–19, 2014.
[36] Y. Wang, X. Liu, H. Wei, G. Forman, C. Chen, and Y. Zhu, "Crowdatlas: Self-updating maps for cloud and personal use," in *Proc. 11th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2013, pp. 469–470.
[37] Y. Wang, Y. Zheng, and Y. Xue, "Travel time estimation of a path using sparse trajectories," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 25–34.
[38] Y. Wang, Y. Zhu, Z. He, Y. Yue, and Q. Li, "Challenges and opportunities in exploiting large-scale GPS probe data," HP Laboratories, Palo Alto, CA, USA, HP Tech. Rep. HPL-2011–109, 2011.
[39] X. Xin, C. Lu, Y. Wang, and H. Huang, "Forecasting collector road speeds under high percentage of missing data," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 1917–1923.
[40] B. Yang, C. Guo, and C. S. Jensen, "Travel cost inference from sparse, spatio-temporally correlated time series using Markov models," *Proc. VLDB Endowment*, vol. 6, pp. 769–780, 2013.
[41] B. Yang, M. Kaul, and C. S. Jensen, "Using incomplete information for complete weight annotation of road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 5, pp. 1267–1279, May 2014.
[42] Z. Yang, J. Hu, Y. Shu, P. Cheng, J. Chen, and T. Moscibroda, "Mobility modeling and prediction in bike-sharing systems," in *ACM Int. Conf. Mobile Syst. Appl. Serv.*, 2016.
[43] J. Yoon, B. Noble, and M. Liu, "Surface street traffic estimation," in *Proc. 5th Int. Conf. Mobile Syst. Appl. Serv.*, 2007, pp. 220–232.
[44] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2012, pp. 2–2.
[45] D. Zhang, J. Huang, Y. Li, F. Zhang, C. Xu, and T. He, "Exploring human mobility with multi-source data at extremely large metropolitan scales," in *Proc. 20th Annu. Int. Conf. Mobile Comput. Netw.*, 2014, pp. 201–212.
[46] D. Zhang, F. Zhang, and T. He, "MultiCalib: National-scale traffic model calibration in real time with multi-source incomplete data," in *Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2016, Art. no. 19.
[47] D. Zhang, J. Zhao, F. Zhang, and T. He, "UrbanCPS: A cyber-physical system based on multi-source big infrastructure data for heterogeneous model integration," in *Proc. ACM/IEEE 6th Int. Conf. Cyber-Phys. Syst.*, 2015, pp. 238–247.
[48] J. Zheng and L. M. Ni, "Time-dependent trajectory regression on road networks via multi-task learning," in *Proc. 27th AAAI Conf. Artif. Intell.*, 2013, pp. 1048–1055.
[49] P. Zhou, S. Jiang, and M. Li, "Urban traffic monitoring with the help of bus riders," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst.*, 2015, pp. 21–30.
[50] P. Zhou, Y. Zheng, and M. Li, "How long to wait?: Predicting bus arrival time with mobile phone based participatory sensing," *Proc. ACM MobiSys*, 2012, pp. 379–392.
[51] H. Zhu, Y. Zhu, M. Li, and L. M. Ni, "SEER: Metropolitan-scale traffic perception based on lossy sensory data," in *Proc. IEEE INFOCOM*, 2009, pp. 217–225.
[52] Y. Zhu, Z. Li, H. Zhu, M. Li, and Q. Zhang, "A compressive sensing approach to urban traffic estimation with probe vehicles," *IEEE Trans. Mobile Comput.*, vol. 12, no. 11, pp. 2289–2302, Nov. 2013.

**Zhidan Liu** received the BE degree in computer science and technology from Northeastern University, China, in 2009, and the PhD degree in computer science and technology from Zhejiang University, China, in 2014. He is currently an assistant professor with Shenzhen University, China. His research interests include distributed sensing and mobile computing, big data analytics, and urban computing. He is a member of the IEEE.

**Pengfei Zhou** received the BE degree from the Automation Department, Tsinghua University, Beijing, China, in 2009, and the PhD degree from the School of Computer Science and Engineering, Nanyang Technological University, Singapore, in 2015. His research interests include mobile computing and systems, localization, and cellular network communications.

**Zhenjiang Li** received the BE degree in computer science and technology from Xian Jiaotong University, Xian, China, in 2007, the MPhil degree in electronic and computer engineering, and the PhD degree in computer science and engineering from The Hong Kong University of Science and Technology, Hong Kong, in 2009 and 2012, respectively. He is currently an assistant professor of the Computer Science Department, City University of Hong Kong. His research interests include wearable and mobile sensing, deep learning and data mining, distributed, and edge computing. He is a member of the IEEE.



**Mo Li** received the BS degree in computer science and technology from Tsinghua University, China, in 2004 and the PhD degree in computer science and engineering from The Hong Kong University of Science and Technology, Hong Kong, in 2009. He is currently an associate professor with Nanyang Technological University, Singapore. His research interests include networked and distributed sensing, wireless and mobile, cyber-physical systems, smart city, and urban computing. He is a member of the IEEE.