Stochastic Weight Completion for Road Networks using Graph Convolutional Networks

Jilin Hu Chenjuan Guo✉ Bin Yang Christian S. Jensen
Department of Computer Science, Aalborg University, Aalbrog, Denmark
{hujilin, cguo, byang, csj}@cs.aau.dk

***Abstract*—Innovations in transportation, such as mobility-on-demand services and autonomous driving, call for high-resolution routing that relies on an accurate representation of travel time throughout the underlying road network. Specifically, the travel time of a road-network edge is modeled as a time-varying distribution that captures the variability of traffic over time and the fact that different drivers may traverse the same edge at the same time at different speeds. Such stochastic weights may be extracted from data sources such as GPS and loop detector data. However, even very large data sources are incapable of covering all edges of a road network at all times. Yet, high-resolution routing needs stochastic weights for all edges.**

We solve the problem of filling in the missing weights. To achieve that, we provide techniques capable of estimating stochastic edge weights for all edges from traffic data that covers only a fraction of all edges. We propose a generic learning framework called Graph Convolutional Weight Completion (GCWC) that exploits the topology of a road network graph and the correlations of weights among adjacent edges to estimate stochastic weights for all edges. Next, we incorporate contextual information into GCWC to further improve accuracy. Empirical studies using loop detector data from a highway toll gate network and GPS data from a large city offer insight into the design properties of GCWC and its effectiveness.

I. Introduction

We are witnessing increasing needs for high-resolution routing that takes into account the dynamics and uncertainty of traffic [1], [2]. For instance, consider a person taking an autonomous taxi to catch a flight. If the taxi takes into account the travel speed distributions of different candidate paths, rather than just average speeds, it is able to choose the path with the highest probability of arriving on time [3]. Using only average speeds often leads to unreliable path choices [4]. Consider an example where two paths P_1 and P_2 lead to the airport. Based on the speed distributions of the edges in the paths, we are able to derive the paths' travel time distributions: P_1 has travel time distribution $\{(30, 0.2), (40, 0.8)\}$, meaning that traversing P_1 may take 30 or 40 mins with probabilities 0.2 and 0.8, respectively; and P_2 has distribution $\{(30, 0.5), (40, 0.3), (50, 0.2)\}$. If the passenger needs to arrive in the airport within 40 mins, taking P_1 is the best since it guarantees an on-time arrival. In contrast, taking P_2 has a 0.2 probability of arriving late. However, if considering only average travel times, P_2 is recommended since its average 37 mins is smaller than that of P_1, i.e., 38 mins.

Such high-resolution routing calls for a road network graph where *every* edge has a time-dependent, stochastic edge weight that captures uncertain traffic dynamics [5], [6]. Various types of traffic data, ranging from GPS data to loop detector data [7], [8], can be used to obtain time-dependent and stochastic edge weights. However, such traffic data often lacks the coverage needed to assign weights to all edges. Loop detectors are typically deployed only on some edges due to high deployment costs; and some loop detectors may be malfunctioning during some periods [9]. Next, a recent study shows that GPS data is often skewed, making it almost impossible to collect sufficient GPS data to cover all edges, during all time intervals [10], [11]. We call this the *data sparseness* problem.

In this paper, we formalize a *stochastic weight completion* problem. Given a traffic data set that only covers a subset of the edges in a road network, the objective is to associate each edge with accurate stochastic weights. Consider the example in Figure 1 where the road network has 6 directed edges. Assume that during [8:15, 8:30], only edges e_5 and e_6 are covered by GPS data and can be associated with stochastic weights in this interval. These weights are represented as travel speed histograms, as shown in the figure. For instance, when traversing edge e_5 during [8:15, 8:30], it may take 5 m/s to 10 m/s with probability 0.3, 10 m/s to 15 m/s with probability 0.5, and 15 m/s to 20 m/s with probability 0.2. The weights for the remaining edges, i.e., e_1, e_2, e_3, and e_4, are missing. During other intervals, different edges may have GPS data and thus can be assigned weights, while the remaining edges cannot.

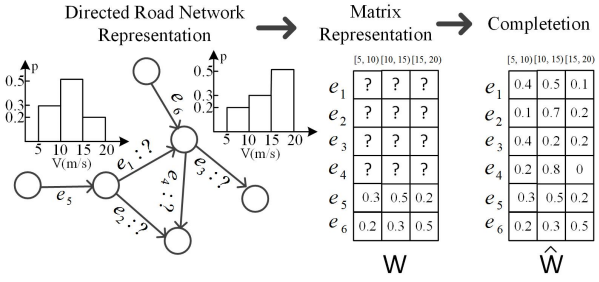


Figure 1: Example of Stochastic Weight Completion.

We convert this information into a matrix representation to facilitate processing, where each row represents the stochastic weight of an edge. For example, the first four rows in W are empty since weights for $e_1, \cdots, e_4$ are missing. The 5-th and 6-th rows represent the stochastic weights of e_5 and e_6. The goal of the stochastic weight completion is to estimate

the stochastic weights for the edges that are not covered by traffic data, i.e., $e_1, \cdots, e_4$. The final result is a new matrix $\hat{W}$, where the empty rows in W are filled with values so that the stochastic weights are available for all edges.

Travel speeds of different edges in a road network exhibit high dependencies. In particular, studies addressing the data sparseness problem often assume that adjacent edges [9], [12], [13] tend to have similar (travel speed based) weights. Thus, the weights of the edges covered by traffic data can be propagated to their adjacent edges that are not covered by traffic data, using regression with judiciously designed loss functions that consider the discrepancies of the weights between the adjacent edges.

However, similarity assumptions may not always be true, since the correlations among the travel speeds of different edges can be very complex. Considering only the weight similarities between adjacent edges is unable to model complex correlations accurately. Further, existing studies only consider deterministic weights (e.g., average travel speeds). It is non-trivial to extend them to support stochastic weights such as travel speed distributions. A data driven approach that is able to capture complex correlations and to support stochastic weights is desirable.

We propose a data-driven, deep learning based framework, with the goal of capturing complex correlations among edge weights in a road network which in turn helps us estimate stochastic weights for edges without data. In particular, we first encode the topology of a road network using spectral graph theory [14] into a graph convolutional neural network (GCNN). Then, we feed available traffic data into the GCNN as both the input and the labeled output to let the GCNN learn complex correlations of edge weights in an "unsupervised" manner, i.e., without requiring additional labelled data as output. The learned GCNN is then employed to complete stochastic weights for the edges without data. Further, we propose an advanced model that takes as input additional context information, e.g., time intervals, day of week, etc., which is able to further improve accuracy of the completed weights. The proposed framework is generic in the sense that it is able to support both stochastic and deterministic edge weights, and it also outperforms the state-of-the-art method when completing deterministic edge weights.

To the best of our knowledge, this is the first study of stochastic weight completion. In particular, we make four contributions. First, we formalize the stochastic weight completion problem. Second, we propose a data-driven framework using a graph convolution neural network to solve this problem. Third, we extend the framework by taking into account additional contextual information, which further improves accuracy. Fourth, we conduct extensive experiments using both GPS and loop detector data sets to provide insight into the effectiveness of the framework.

The remainder of the paper is organized as follows. Section II covers related work. Section III defines the setting and formalizes the problem. Section IV and Section V detail the framework of graph convolutional weight completion (GCWC) and context aware graph convolutional weight completion (A-GCWC), respectively. Section VI reports experiments and results. Section VII concludes.

II. Related Work

Data Spareness in Road Networks: Although traffic prediction has been studied extensively [15], [16], only a few studies [9], [12], [13], [17] consider the data sparseness problem in road networks. The basic ideas of the *trajectory regression* problem [12], [13], [17] have been covered in Section I. More recently, a latent space model (LSM) is proposed to estimate the weights of edges that are not covered by loop detector data [9]. Non-negative matrix factorization is used as an encoder to learn the latent space features, which helps estimate the weights of edges without data. LSM is the state-of-the-art method.

All existing studies that address the data sparseness problem only consider deterministic weights and cannot be extended to support stochastic weights in a straightforward manner. In addition, they all employ linear models to cope with correlations among edge weights. However, such correlations can be highly non-linear [18]. We propose a graph convolutional weight completion framework that enables stochastic weight annotation while considering non-linear weight correlations.

Deep Learning in Transportation: RNNs with auto-encoders are proposed to enable traffic forecasts using traffic sensor data [18]. However, this proposal ignores spatial correlations among the sensors. To address this problem, another method uses diffusion convolutional networks, which are able to model spatial correlations, together with RNNs to enable traffic forecasts [19]. Alternatively, classic convolutional networks can also model sensor correlations [20]. However, these methods are restricted to deterministic traffic values and do not support stochastic values. In addition, it is assumed that sufficient traffic data is available to cover all edges in a road network while our proposal considers the case when data is sparse. A more recent study focuses on travel time estimation for origin-destination pairs, but not for edges [21], using multi-task learning. Multi-task learning is also applied to distinguish trajectories from different drivers [22]. To the best of our knowledge, this paper proposes the first deep learning framework for stochastic weight completion in road network graphs.

III. Preliminaries and Problem Formulation

A. Road Network

A road network is often represented as a directed graph $H = (V, E)$, where vertex set V represents road intersections and edge set $E \subseteq V \times V$ represents directed edges. We model a road network as an edge graph $G = (E, A)$, where E is the edge set, and A is an $|E| \times |E|$ adjacency matrix that captures how the directed edges are connected. In particular, $A_{i,j} = 1$ if travel is possible from edge e_i to edge e_j or from edge e_j to edge e_i via a single vertex; otherwise, $A_{i,j} = 0$. This makes matrix A symmetric and the edge graph undirected.

Figure 2 shows a road network and its corresponding edge graph and adjacency matrix. For example, $A_{5,2} = 1$ because

a vehicle can travel from e_5 to e_2 by traversing one vertex in the road network. However, since travel from e_2 to e_1 and travel from e_1 to e_2 via a single vertex are not possible, we have $A_{2,1} = 0$.

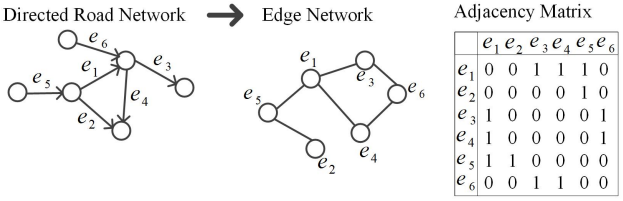


Figure 2: Road Network and Its Edge Graph.

B. Stochastic Weights

To capture time-dependent traffic, we partition a day into a number of intervals, e.g., 96 15-min intervals. Based on this, we introduce a stochastic weight function $\mathcal{F} : E \times TI \rightarrow D$, where TI is the whole time domain of interest and D is a set of all possible speed distributions. Given a specific edge $e_i \in E$ and a time interval $T_j \in TI$, function $\mathcal{F}(e_i, T_j)$ returns a stochastic weight that represents the speed distribution on edge e_i during interval T_j.

To instantiate the weight function $\mathcal{F}$, we need to assign stochastic weights to all edges for each interval $T_j \in TI$. In the following discussion, we focus on instantiating $\mathcal{F}$ for a specific interval T_j.

We first identify the traffic data available in T_j. Next, we partition all edges into subsets E_c and E_m, the edges with and without traffic data, respectively. This means that $E_c \cup E_m = E$ and $E_c \cap E_m = \emptyset$. In Figure 1, we have $E_c = \{e_5, e_6\}$ and $E_m = \{e_1, e_2, e_3, e_4\}$ for interval [8:15, 8:30].

For each edge $e_c \in E_c$, which is covered by traffic data, we are able to derive a stochastic weight and thus able to instantiate $\mathcal{F}(e_c, T_j)$. However, we are unable to instantiate $\mathcal{F}(e_m, T_j)$ if edge $e_m \in E_m$. For example, during [8:15, 8:30], GPS trajectories exist for edges e_5 and e_6, and thus we are able to use them to build speed distributions, i.e., stochastic weights for e_5 and e_6, during [8:15, 8:30], thus instantiating $\mathcal{F}(e_5$, [8:15, 8:30]) and $\mathcal{F}(e_6$, [8:15, 8:30]).

We use equi-width histograms to represent speed distributions. In particular, an equi-width histogram is a set of bucket-probability pairs $\{(b_i, p_i)\}$. A bucket $b_i = [l_i, u_i)$ represents the speed range from l_i to u_i, and all buckets have the same range size. Next, p_i is the probability that the speed falls into range b_i. For example, the speed histogram $\{([0, 20), 0.5)$, $([20, 40), 0.3)$, $([40, 60), 0.2)\}$ for edge e_5 means that the probability that the speed (m/s) on e_5 falls into $[0, 20)$, $[20, 40)$, and $[40, 60)$ is 0.5, 0.3, and 0.2, respectively.

We use the same finest bucket range size for all edges' speed histograms. Thus, we can ignore the buckets and represent the speed histogram of each edge as a vector. For example, when choosing 20 m/s as the bucket range size, e_5's speed histogram can be represented as $\langle 0.5, 0.3, 0.2 \rangle$.

Assume that we have $n = |E|$ edges in the road network and we use a histogram with m buckets to present a stochastic weight. Then, the stochastic weights of all edges in interval T_j can be represented as an $n \times m$ matrix W. A row vector $w_{i\cdot}$ in W corresponds to the vector representation of edge e_i's stochastic weight, i.e., its speed histogram. If an edge $e_i \in E_m$, $w_{i\cdot}$ is an empty vector. For example, the first four rows in W in Figure 1 are empty vectors.

C. Problem Formulation

Consider a time interval T_j of a day. Given the instantiated stochastic weight matrix W, the *stochastic weight completion* problem is that of completing the empty rows in W to produce a new stochastic weight matrix $\widehat{W}$ without empty rows. This is equivalent to instantiating $\mathcal{F}(e_m, T_j)$ for each edge $e_m \in E_m$ that is not covered by traffic data.

To ease the presentation, Table I lists important notation that we use throughout this paper.

Notations	Definition
G	Edge graph of a road network
E, V	Edge set, Vertex set
A	Adjacency matrix of G
n	Total number of edges
m	Total number of buckets in a histogram
W	Input Weight Matrix
W_G	Ground Truth Weight Matrix
$\widehat{W}$, $\widetilde{W}$	Estimated, complete weight matrix
X_i	A context variable

Table I: Notation.

D. Solution Overview

We propose a basic model and an advanced model to solve the problem. The basic model takes as input an instantiated, *incomplete* stochastic weight matrix W and the edge adjacency matrix A. The basic model is able to learn correlated edge features from W and A using graph convolution filters and thus derives a *complete* stochastic weight matrix $\widehat{W}$.

However, the basic model does not make use of important *contextual* information, e.g., time intervals and the day of the week. For example, traffic in peak vs. off-peak intervals may be different and traffic on weekdays vs. weekends may also be different. To better utilize such contexts that are missing in the basic model, but may be useful for completing weights, we propose the advanced model. This model also takes as input the available contexts and applies a Bayesian inference model to construct dependency relationships between contextual features and the output of the basic model with the goal of improving the accuracy of the complete $\widehat{W}$.

We present the basic and the advanced models in Sections IV and V, respectively.

IV. Graph Convolutional Weight Completion

A. Intuitions and Framework Overview

Since traffic on one edge may influence traffic on many other edges [9], [17], it is intuitive to assume that stochastic weights of different edges share correlated features. We model such features by transform stochastic weight matrix W into a set $\mathbb{C} = \{C_i\}_{i=1}^{f}$ of latent variables that captures correlations among the weights of edges. Based on the latent variables in

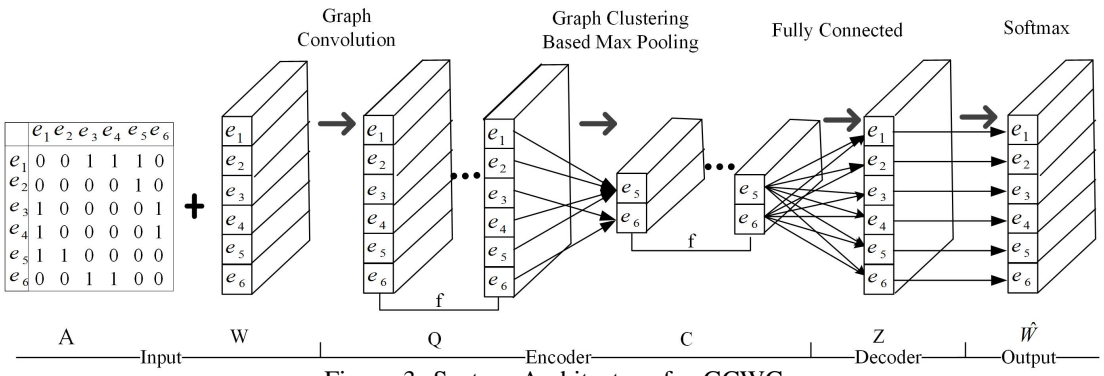


Figure 3: System Architecture for GCWC.

$\mathbb{C}$, we construct a new stochastic weight matrix $\widehat{W}$ without empty rows. The whole process can be regarded as an *auto-encoder* [23], [24], where we first *encode* incomplete weight matrix W into a set of features $\mathbb{C}$ in a latent space and then *decode* $\mathbb{C}$ back to a complete weight matrix $\widehat{W}$.

Figure 3 shows an overview of the basic model for Graph Convolutional Weight Completion, denoted as GCWC, where we adopt the intuition of the auto-encoder.

We provide stochastic weight matrix W and adjacency matrix A to GCWC as input. Next, we use convolutional and max pooling layers to encode W into a set of features $\mathbb{C}$, which can be regarded as the encoding process. Finally, we map the encoded features $\mathbb{C}$ to the final output layer with the help of a fully connected layer and thus obtain an estimated weight matrix $\widehat{W}$, where each edge has a stochastic weight. This corresponds to the decoding process.

In the training phase, input matrix W is also used as a source of labels for conducting back-propagation. We learn the parameters of our framework with the objective of minimizing a loss function that is defined based on the KL-divergence between the estimated stochastic weights and the ground truth stochastic weights, i.e., the instantiated stochastic weights of the edges that are covered by traffic data (details to be provided in Section IV-E).

B. Convolutional Layer

In classical convolutional neural networks (CNNs), 2D convolutional filters are applied in convolutional layers based on the assumption that nearby elements in the input matrix share local features [14]. For example, when representing an image as a matrix, nearby elements, e.g., pixels, share local features, e.g., represent parts of the same object. However, in our setting, the input stochastic weight matrix may not always satisfy this assumption—two adjacent rows in matrix W may represent two geometrically distant road network edges and may not share any features. In Figure 1, although the rows for e_5 and e_6 are adjacent in W, e_5 and e_6 are not adjacent in the road network. This renders classical 2D convolutional filters ineffective in our setting and calls instead for new filters that take into account the topology of the road network, e.g., through the use of adjacency matrix A. Our solution utilizes recently invented graph convolutional neural networks (GCNNs) [14], [25].

Background on GCNNs: In GCNNs, *graph convolutional filters* [14], [25], [26], which take into account the topology of a road network based on spectral graph theory, are employed to replace the classic 2D convolutional filters in the convolutional layers. Graph convolutional filters consider that topologically adjacent elements in a graph share local features. By using graph convolutional filters, we can "propagate" the input stochastic weights to adjacent, correlated edges during convolutions via the road network topology. In the literature, different variations of graph convolutional filters exist. We use *Simplified ChebNet* [26], due to its efficiency and effectiveness.

Simplified ChebNet: Since graph convolutional filters are based on spectral graph theory, the central component of a graph filter is the graph Laplacian [14], [26]. To construct the graph Laplacian L, we utilize adjacency matrix A that captures the topology of the edge graph of a road network (see Section III). In particular, $L = D - A$, where D is the diagonal degree matrix with $D_{i,i} = \Sigma_j A_{i,j}$. Next, we derive the scaled Laplacian $\tilde{L} = 2L/\lambda_{max} - I$, where λ_{max} is the maximum eigenvalue of L and I is an identify matrix.

In the convolution layer, we apply graph convolutional filters to the input stochastic weight matrix W by using scaled Laplacian $\tilde{L}$. Recall that column vector $w_{\cdot j} \in \mathbb{R}^{n\times 1}$ of the input stochastic weight matrix W represents the weights of all n edges in the j-th bucket (where $j \in [1, m]$). Based on $w_{\cdot j}$ and $\tilde{L}$, we first generate a matrix $Y_j \in \mathbb{R}^{n\times k}$ and $Y_j = [\hat{x}_0, \hat{x}_1, \cdots, \hat{x}_{k-1}]$, where $\hat{x}_i \in R^{n\times 1}$ is a column vector and k is a hyper-parameter. Specifically, we have $\hat{x}_0 = w_{\cdot j}$, i.e., the original input column vector. Next, $\hat{x}_1 = \tilde{L}\hat{x}_0$. When $k \geq 2$, $\hat{x}_k$ is defined recursively: $\hat{x}_k = 2\tilde{L}\hat{x}_{k-1} - \hat{x}_{k-2}$. Further, we define a filter τ as an $\mathbb{R}^{k\times 1}$ matrix, and we use a total of f filters.

Based on the above, the convolution using graph convolutional filters on an input vector $w_{\cdot j}$ is defined based on τ and Y_j using Equation 1.

$$H_j = \tanh(Y_j \cdot \tau + b) = \tanh([\hat{x}_0, \hat{x}_1, \cdots, \hat{x}_{k-1}] \cdot \tau + b) \quad (1)$$

Here, $\cdot$ denotes matrix multiplication, and thus we have $H_j \in \mathbb{R}^{n\times 1}$.

The example in Figure 1 features 6 edges and 3 buckets, which yields the first column vector $w_{\cdot 1} = [0, 0, 0, 0, 0.3, 0.2]^T$ for the $[5, 10)$ bucket (i.e., the first bucket). Assuming $k = 2$, we construct the corresponding Y_1 from adjacency matrix A as

$\begin{pmatrix} 0 & 0 & 0 & 0 & 0.3 & 0.2 \\ -0.3 & -0.3 & -0.2 & -0.2 & 0.3 & 0.2 \end{pmatrix}^T$. Next, we apply a filter $\tau = [1, -1]^T$ to Y_1 and get $H_1 = [0.3, 0.3, 0.2, 0.2, 0, 0]^T$. Thus, data for edges e_5 and e_6 are propagated to edges $e_1, \cdots, e_4$ by means of graph convolution.

Given an input stochastic weight matrix W, we apply all f filters, $\tau_1, \ldots, \tau_f$, to each column vector $w_{\cdot j}$ in W, $j \in [1, m]$. Thus, for each column vector $w_{\cdot j}$, we obtain its f corresponding matrices $H_j^1, \ldots, H_j^f$. Next, for each filter τ_l, $1 \leq l \leq f$, we sum the convoluted results of all column vectors $w_{\cdot j}$, $1 \leq j \leq m$ as an $\mathbb{R}^{n \times 1}$ vector $Q_l = \sum_{j=1}^{m} H_j^l$. Next, we concatenate a total of f vectors as an $\mathbb{R}^{n \times f}$ matrix $Q = [Q_1, \ldots, Q_f]$, as the final result from the convolutional layer. Referring back to the auto-encoder model, here the f filters will eventually construct a set of f features in the latent space.

C. Pooling Layer

Although the convolutional layer tries to propagate stochastic weights to the edges that are not covered by traffic data, some edges may still have zero values. Therefore, we further compress the convoluted results, i.e., Q_1, Q_2, ..., Q_f, via max-pooling layers, which follows the design principles of traditional CNNs. Specifically, we employ a multi-level graph-based pooling algorithm [14] to first identify clusters of edges using the graph topology and distributions of available stochastic weights, and then the identified clusters are used as pools to perform max-pooling operations. For example, Figure 3 shows an example where e_1, e_2, e_4, e_5 are clustered into one pool and e_3 and e_6 are clustered into another pool. Based on the pools, each convoluted matrix Q_l is further compressed into a more compact matrix C_l, where $1 \leq l \leq f$. Now, we regard such compact matrices as the feature set $\mathbb{C} = \{C_1, C_2, \ldots, C_f\}$ in the latent space.

D. Output Layer

After pooling, we obtain compact matrices that capture representative features of input matrix W. Next, we perform decoding that uses the compact matrices to produce a new stochastic weight matrix $\widehat{W}$. We first utilize a fully-connected (FC) layer to obtain a matrix Z, representing stochastic weights of all edges decoded from feature set $\mathbb{C}$. In particular, $Z = [z_{1\cdot}, \cdots, z_{n\cdot}]^T$, where n is the total number of edges, and $z_{i\cdot} \in \mathbb{R}^m$, with m being the number of buckets in histograms, is used to estimate the stochastic weight vector for the i-th edge.

The final output $\widehat{W} \in \mathbb{R}^{n \times m}$ must meet two requirements to be a meaningful histogram: (1) for each value $\widehat{w}_{i,j}$ of $\widehat{W}$, we have $0 \leq \widehat{w}_{i,j} \leq 1$, with $1 \leq i \leq n$ and $1 \leq j \leq m$; (2) $\sum_j \widehat{w}_{i,j} = 1$, meaning that the sum of values in a histogram for the i-th edge equals 1, thus aligning with our definition of stochastic weights (see Section III-B).

To this end, a softmax function is applied to every $z_{i\cdot}$:

$$\widehat{w}_{i\cdot} = softmax(z_{i\cdot}), 1 \leq i \leq n, \tag{2}$$

where $\widehat{w}_{i\cdot}$ is the estimated histogram for the i-th edge. Thus, we have $\widehat{W} = [\widehat{w}_{1\cdot}, \cdots, \widehat{w}_{n\cdot}]^T$.

E. Loss Function

The loss function $L(W, \widehat{W})$ of GCWC measures the discrepancy between the input stochastic weight matrix $W \in \mathbb{R}^{n \times m}$ and the estimated stochastic weight matrix $\widehat{W} \in \mathbb{R}^{n \times m}$ using KL-divergence, as shown in Equation 3.

$$L(W, \widehat{W}) = \sum_{i=1}^{n} I_i \cdot KL(w_{i\cdot} || \widehat{w}_{i\cdot}), \text{where}$$
$$KL(w_{i\cdot} || \widehat{w}_{i\cdot}) = \sum_{j=1}^{m} \widehat{w}_{ij} \cdot \log \frac{\widehat{w}_{ij} + \epsilon}{w_{ij} + \epsilon}. \tag{3}$$

Specifically, we have n edges in total and the function focuses on the edges that are covered by traffic data, i.e., the non-empty rows in W, since we use the loss function to measure whether the estimated weights are similar to the original weights. Thus, we apply an indicator I_i, and we set $I_i = 1$ if the i-th edge is covered by traffic data; otherwise, we set $I_i = 0$.

Then, the KL-divergence measures the divergence of the estimated weight $\widehat{w}_{i\cdot}$ from the actual stochastic weight $w_{i\cdot}$ on the i-th edge, where w_{ij} and $\widehat{w}_{ij}$ are the actual and estimated weights for the i-th edge at the j-th bucket. In total, we have m buckets. We apply a positive small value ϵ to prevent having a zero when using the *log* function.

V. Context Aware Graph Convolutional Weight Completion

A. Context Aware Graph Convolution Neural Network

We proceed to present how contexts can be integrated into GCWC to enable an advanced model A-GCWC, as presented in Figure 4.

When we consider stochastic weight matrix W that is instantiated during time interval T_j of a day, we can consider a set $\mathbb{X}$ of contexts. Specifically, we consider three contexts—time interval X_T, day of the week X_D, and row flag X_R.

Assume that a day is partitioned into 96 15-min intervals. We are able to use a one-hot vector with 96 bits to represent a specific time interval. For example, if a weight matrix W is instantiated during [0:15, 0:30], the 2-nd bit in the vector is set to 1 and all remaining bits are set to 0. This vector is used as the time interval context X_T. Next, we use a 7-bit vector for X_D to represent the week days. Finally, the row flag X_R of a weight matrix W indicates the non-empty rows. In particular, X_R is a vector with n bits, where n is the number of edges. If the k-th edge is not empty, the k-th bit in X_R is set to 1; otherwise, it is set to 0.

Figure 4 gives an overview of the A-GCWC framework, which consists of a basic *GCWC*, a *context embedding* module, and a *Bayesian inference* module.

The basic GCWC takes as input an incomplete weight matrix W and an adjacency matrix A, and it outputs an estimated complete weight matrix. The intuition is to learn stochastic weights of all edges without considering the contexts. Recall that in GCWC, we apply a softmax function on the output of the fully connected layer, i.e., Z, to produce a valid weight matrix $\widehat{W}$. Here, we use $P(Z)$ to denote the output of GCWC $\widehat{W}$ where $P(\cdot)$ denotes the softmax function (see

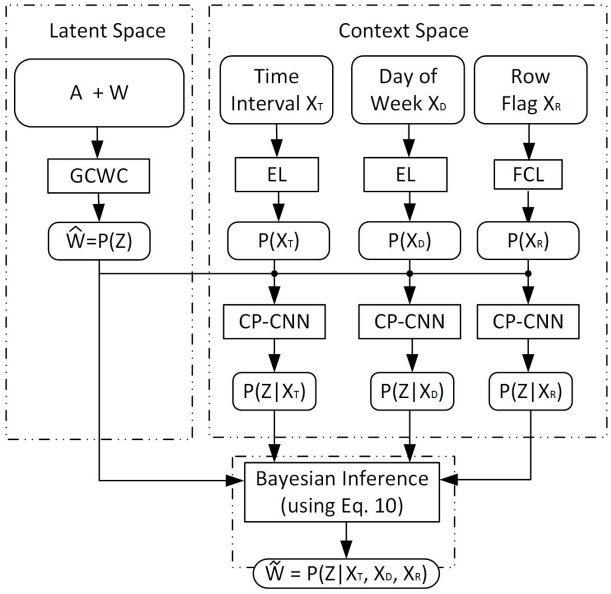


Figure 4: Context Aware Graph Convolution Neural Network.

Section IV-D), which can also be interpreted as a probability function.

The context embedding module first embeds the provided contexts X_T, X_D, and X_R into a space with the same dimensionality. Then, together with the output from the GCWC, i.e., $P(Z)$, it represents the conditional probability of having weight matrix Z conditioned on each context, denoted as $P(Z|X_T)$, $P(Z|X_D)$, and $P(Z|X_R)$, respectively.

The Bayesian inference module takes as input $P(Z)$ from the GCWC, and $P(Z|X_T)$, $P(Z|X_D)$, and $P(Z|X_R)$ from the context embedding module, and it infers the conditional probability for all contexts. Thus, more accurate stochastic weights given all types of context can be learned as $P(Z|X_T, X_D, X_R)$. We denote the estimated weight matrix from A-GCWC as $\widetilde{W} = P(Z|X_T, X_D, X_R)$, which is different from the estimated matrix from the basic GCWC, $\widehat{W} = P(Z)$.

B. Context Embedding Module

The proposed context embedding module is generic—although we only use three types of context, the module is extendable to include, e.g., weather conditions, wind speeds and traffic flows, that may further improve the overall accuracy, if they are available.

Due to different representations of X_T, X_D, and X_R, we apply two different models, i.e., an embedding layer and a fully connected layer, to incorporate the different types of contexts into A-GCWC.

1) Embedding Layer: We apply an embedding layer (EL) for time interval context X_T and day of the week context X_D, since both are represented as one-hot vectors. The embedding method [27] was initially proposed in order to effectively transform a categorical value represented by a high-dimensional, one-hot vector into a low-dimensional vector. As a result, neural network can process the categorized value more efficiently.

For generality, we assume that a total of α time intervals are considered. Thus, we have vector $X_T \in \mathbb{R}^{\alpha \times 1}$ as the one-hot representation for the time interval. The embedding layer is able to transform X_T into $\widehat{X}_T \in \mathbb{R}^{\beta \times 1}$ where $\beta \ll \alpha$. Next, we apply the softmax activation function to compute a distribution for X_T using $\widehat{X}_T$, denoted as $P(X_T)$. Specifically, $P(X_T)$ represents the probability of having each embedded context value in $\widehat{X}_T$. Similarly, we are able to derive $P(X_D)$.

2) Fully Connected Layer: A row flag vector X_R may have more than one occurrence of "1". Since some embedding methods, especially those based on lookup tables [28], require inputs represented as one-hot representations, we instead apply a fully connected layer (FCL) to embed $X_R \in \mathbb{R}^{n \times 1}$ into a smaller space $\widehat{X}_R \in \mathbb{R}^{\beta \times 1}$, where $\beta \ll n$, as follows.

$$P(X_R) = \sigma(\widehat{X}_R) \text{ where } \widehat{X}_R = M \times X_R + b, \qquad (4)$$

where $M \in \mathbb{R}^{\beta \times n}$ is a weight matrix, $b \in \mathbb{R}^{\beta \times 1}$ is a bias vector, and σ is a softmax function. Based on the above, $P(X_R)$ represents the probability of each value occuring in $\widehat{X}_R$.

3) Computing Conditional Probabilities: We utilize a conditional probability convolutional neural network (CP-CNN) to capture the dependency between the stochastic weight of an edge $z_{j\cdot}$ and each type of context, as shown in Figure 5. To simplify our discussion, we use X_i to denote a context, where X_i can be X_T, X_D, or X_R, in the following.

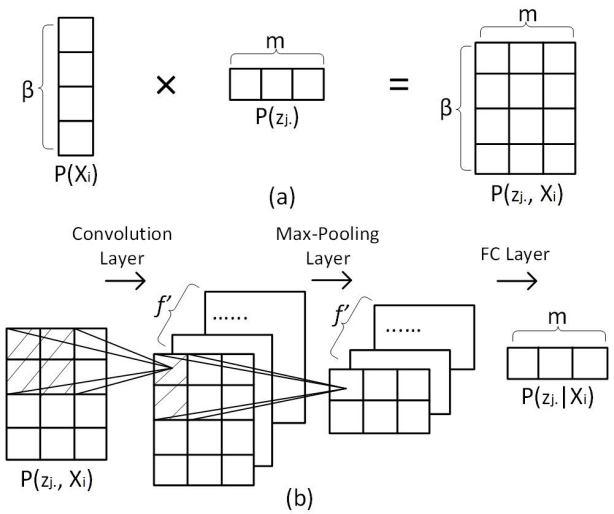


Figure 5: Conditional Probability CNN (CP-CNN).

As shown in Figure 5(a), we multiply $P(X_i)$ with $P(z_{j\cdot})$, where $P(X_i) \in \mathbb{R}^{\beta \times 1}$ is the probability distribution of the embedded context values for context X_i and $P(z_{j\cdot}) \in \mathbb{R}^{1 \times m}$ is the estimated stochastic weight for the j-th edge obtained by GCWC, with m being the bucket size. As a result, we obtain a matrix, denoted as $P(z_{j\cdot}, X_i) \in \mathbb{R}^{\beta \times m}$, that associates each bucket of the j-th edge's weight with each embedded context value in context X_i, based on which we learn whether a bucket and an embedded context value exhibit a dependency.

Next, we capture such possible dependencies using classical convolutional filters. Following the running example in Figure 1, we have $m = 3$ buckets as $[0, 20)$, $[20, 40)$, and $[40, 60)$.

If we apply a filter of size 2×2 to the 4 shadowed squares in the leftmost matrix of Figure 5(b), we are able to capture the dependency between 2 buckets, e.g., $[0, 20)$, $[20, 40)$, and 2 values of context X_i. This is intuitive, since the probabilities of speeds falling into $[0, 20)$ and $[20, 40)$ influence each other, and they may also be influenced by similar contexts, e.g., intervals [8:00, 8:15] and [8:15, 8:30].

As shown in Figure 5(b), we utilize f' filters and obtain a total of f' matrices, each of the same sizes as $\mathbb{R}^{\beta \times m}$. Next, a classical max-pooling layer with window size 2 is applied to learn more representative dependencies, giving rise to a total of f' matrices, each of the same size as $\mathbb{R}^{\frac{\beta}{2} \times m}$. We apply a fully connected layer to concatenate f' matrices into an $\mathbb{R}^{1 \times m}$ matrix as the conditional probability between the stochastic weight of the j-th edge and context X_i, denoted as $P(z_j|X_i)$.

After we conduct the same procedure for all n edges, we obtain $P(Z|X_i)$, i.e., the weight matrix when considering context X_i.

C. Bayesian Inference

We utilize the Bayesian Inference module to derive a probability distribution for weight matrix Z given all types of contexts X_T, X_D, and X_R.

For generality, we assume that we have N types of contexts $X_1, \ldots, X_N$ and that we have obtained $P(Z|X_1)$, ..., $P(Z|X_N)$ as the conditional probability of Z given each context X_i, where $i \in [1, N]$ (cf. Section V-B3). Further, we have obtained $P(Z)$ from the basic GCWC. We aim to infer $P(Z|X_1, \ldots, X_N)$ as the stochastic weight $\widetilde{W}$, i.e., the conditional probability of Z given all contexts $X_1, \ldots, X_N$, from $P(Z|X_1)$, ..., $P(Z|X_N)$ and $P(Z)$.

To this end, we make the assumption that different contexts are independent, and thus we have $P(X_1, \ldots, X_N) = \prod_{i=1}^{N} P(X_i)$. This is a reasonable assumption because, for example, time intervals, day of week, and row flags do not have obvious correlations. Then, we have

$$P(Z|X_1, \ldots, X_N) \tag{5}$$

$$= \frac{P(Z, X_1, \ldots, X_N)}{P(X_1, \ldots, X_N)} = \frac{P(Z, X_1, \ldots, X_N)}{\prod_{i=1}^{N} P(X_i)} \tag{6}$$

$$= \frac{P(Z)P(X_1, \ldots, X_N|Z)}{\prod_{i=1}^{N} P(X_i)} = \frac{P(Z)\prod_{i=1}^{N} P(X_i|Z)}{\prod_{i=1}^{N} P(X_i)} \tag{7}$$

$$= \frac{\prod_{i=1}^{N}(P(X_i|Z)P(Z))}{[P(Z)]^{(N-1)}\prod_{i=1}^{N} P(X_i)} = \frac{\prod_{i=1}^{N} \frac{P(X_i,Z)}{P(X_i)}}{[P(Z)]^{(N-1)}} \tag{8}$$

$$= \frac{\prod_{i=1}^{N} P(Z|X_i)}{[P(Z)]^{(N-1)}} \tag{9}$$

Here we keep using Bayesian rule and the independence assumption in Equations 6, 7, and 8.

According to Equation (9), we compute $\widetilde{W} = P(Z|X_T, X_D, X_R)$ using Equation (10).

$$P(Z|X_T, X_D, X_R) = \frac{P(Z|X_T)P(Z|X_D)P(Z|X_R)}{[P(Z)]^2}, \tag{10}$$

where $P(Z|X_T, X_D, X_R)$ is the estimated stochastic weight given context X_T, X_D, and X_R, which is further normalized to get $\widetilde{W}$. The normalization makes sure that: (1) for each $\widetilde{w}_{i,j} \in \widetilde{W}$, $0 \leq \widetilde{w}_{i,j} \leq 1$, with $1 \leq i \leq n$ and $1 \leq j \leq m$; (2) $\sum_j \widetilde{w}_{i,j} = 1$.

D. Loss Function

The loss function for A-GCWC is based on KL-divergence—we compute $L(W, \widetilde{W})$ using Equation 3.

VI. Experiments

A. Experimental Setup

1) Data sets: We use two traffic data sets for studying the effectiveness of the proposed models. In both data sets, we report results on the setting of HIST-8, where a histogram has 8 buckets. In particular, the bucket length is 5 m/s, and the range is from 0 to 40 m/s, yielding a total of 8 buckets. Due to the space limitation, we do not report results for the setting of HIST-4 where 4 buckets are used. This setting yields similar results as the HIST-8 setting.

Highway Tollgates Network (HW): *HW* consists of 1.07 million travel time records from loop detectors located in $n = 24$ links in a highway tollgate network. The records cover the period from 19/07/2016 to 31/10/2016. This data is obtained from a big data challenge. We partition a day into 96 15-min intervals. Given the $m = 8$ buckets used in HIST-8, each input matrix $W \in \mathbb{R}^{n \times m}$, i.e., the stochastic weights, is an $\mathbb{R}^{24 \times 8}$ matrix, and we have a total of 96 matrices per day. For each matrix W, we use the corresponding time interval, day of the week, and row flag vector of W as the contexts of the matrix.

City Road Network (CI): *CI* consists of 1.4 billion GPS records from 14,864 taxis obtained in the period from 03/08/2014 to 30/08/2014 in Chengdu, China. This data is obtained from a big data competition. To get reliable results, we use $n = 172$ connected road segments with sufficient GPS data in the experiments. In particular, we choose a dense subgraph with 172 edges where almost all edges have GPS data in most time intervals. This design decision is made because we need to ensure that each edge has sufficient data to derive ground truth weights, such that we can evaluate the accuracy of the estimated weights. Specifically, 1) we select edges with the top-5000 largest amounts of GPS records; 2) we derive the connected subgraphs using the 5000 edges; 3) we use the largest connected subgraph with 172 edges in the experiments.

Similarly, we have 96 time intervals per day in *CI*. Each interval has a stochastic weight matrices $W \in \mathbb{R}^{172 \times 8}$, which is associated with three types of context.

2) Ground Truth and Input Data: We first introduce ground truth weight matrix W_G and then show how we generate input matrix W from W_G.

Given a time interval, we are able to instantiate a ground truth matrix W_G from the available traffic data. We only instantiate weights for edges with at least 5 speed records. Next, we randomly select $n \times rm$ edges from a total of n

edges, where rm is a removal ratio. We set the stochastic weights of the chosen edges in W_G to be zeros, giving rise to the incomplete input matrix W. We use four values for rm: 0.5, 0.6, 0.7, and 0.8. Taking W as input, we use a method to estimate a complete weight matrix $\widehat{W}$ that covers all edges. Then we evaluate the accuracy of $\widehat{W}$ by comparing $\widehat{W}$ with the ground truth matrix W_G.

Note that the matrix W_G may already have empty rows for some edges since the available traffic data in an interval may not cover these edges. Although there are edges with empty rows, we do not estimate a complete weight matrix based on W_G directly. If we simply did so, although we would be able to fill in weights for the edges without data, we are not able to evaluate the accuracy of the estimated weights. This is why we instead remove $n \times rm$ edges' weights from W_G to create input matrices.

For each data set, we order all input matrices in ascending time order, and we partition these into 5 equal-sized portions. Then we use 5-fold cross validation in all the experiments—in each run, 4 folds of matrices are used for training and validating, and the remaining 1 fold of matrices are used for testing. We run this workload 5 times in total, so that each fold of matrices is used for testing once.

3) Model Functionalities: The proposed models, GCWC and A-GCWC, are generic and extendable, and they are able to support different functionalities when changing the configurations slightly. Here, we consider three different functionalities: *estimation* and *prediction* of stochastic weights, in the form of histograms, and estimation of *average* speeds, in the form of deterministic values. We use $W@T$ to denote weight matrix W during interval T.

Estimation: Given input matrix $W@T_i$ that represents stochastic weights at time interval T_i, where some edges do not have weights, we estimate $\widehat{W}@T_i$ that has the estimated stochastic weights at T_i for all edges.

During training, we have a set of training matrices $\{W@T_k\}$. For each training matrix in $\{W@T_k\}$, we use the matrix itself as a label to train the two models. During testing, given input matrix $W@T_i$, the estimated $\widehat{W}@T_i$ is compared with the ground truth matrix at T_i, i.e., $\widehat{W}_G@T_i$, to evaluate the accuracy.

Prediction: Given input matrix $W@T_i$ that represents stochastic weights at time interval T_i, where some edges do not have weights, we predict $\widehat{W}@T_{i+1}$ that contains the predicted stochastic weights in the next time interval T_{i+1} for all edges.

During training, we have a set of training matrices $\{W@T_k\}$. For each training matrix $W@T_k$, we use the ground truth matrix in the next interval, i.e., $W_G@T_{k+1}$, as a label to train the two models. We make sure that $W_G@T_{k+1}$ has the same rm as the input matrix $W@T_k$. For example, when rm is 0.6, both the input matrix $W@T_k$ and label matrix $W_G@T_{k+1}$ have 60% empty edges. During testing, given input matrix $W@T_i$, the estimated $\widehat{W}@T_{i+1}$ is compared with the ground truth weight matrix at T_{i+1}, i.e., $\widehat{W}_G@T_{i+1}$, to evaluate the accuracy. Table II summarizes the settings of estimation and prediction.

	Training		Testing	
	Input	Label	Input	Ground Truth
Estimation	$W@T_k$	$W@T_k$	$W@T_i$	$W_G@T_i$
Prediction	$W@T_k$	$W_G@T_{k+1}$	$W@T_i$	$W_G@T_{i+1}$

Table II: Settings, Estimation vs. Prediction.

Average: This setting is similar to *Estimation*. Given input matrix $W@T_j$, we estimate a deterministic average speed value for each edge during the same interval T_j, rather than a speed histogram. To achieve this, we replace the softmax function in Equation (2) of Section IV-D with the sigmoid function and thus obtain $P(Z) \in \mathbb{R}^{n\times 1}$ in the latent space (see Figure 4). We also replace the normalization on $P(Z|X_T, X_D, X_R)$ in Equation (10) of Section V-C with the sigmoid function so that the output $\widehat{W} \in \mathbb{R}^{n\times 1}$ represents the estimated average speeds for all edges in time interval T_j. The ground truth matrix $W_G@T_j$ of average speeds during T_j is derived by averaging all speed records for each edge.

4) Model Settings: Model Construction: We present hyperparameters of all models in Table III. We refer to the models constructed for the estimation and prediction of speed histograms as being of type HIST, and for the average as being of type AVG. For both types, we describe the hyperparameters used for GCWC and A-GCWC, including the learning rate (LR), learning rate decay (Decay), Dropout, and regularization (Regul). The column with header "#Para" indicates the total number of parameters used in a deep learning model (e.g., parameters for convolution filters, full-connected layers, biases used in activation functions, etc.). This reflects the complexity of different neural networks. The higher the value is, the more complex the corresponding neural network is. The #Para column in Table III shows that the total number of parameters in CNN, GCWC, and A-GCWC are similar, meaning that, compared to classical CNN, the proposed GCWC and A-GCWC do not significantly increase the model complexity.

We use the following notation to describe the model construction: $C_f^{k_1\times k_2}$ denotes a convolution layer that has f filters, each of which is a $k_1 \times k_2$ matrix; P_k denotes a pooling layer of size and stride k; FC_k denotes a fully connected layer with k hidden units. For example, GCWC for HW, HIST is constructed as $C_{16}^{8\times 1}$-P_4-$C_{16}^{8\times 1}$-P_2-FC_n, where n is the number of edges, which varies across data sets.

Further, β, the dimension of the embedded context space, is set to 4 for A-GCWC for all data sets.

We obtain an optimal set of hyperparameters with Bayesian optimization using Gaussian Processes search.

Model Complexity: The time complexity of the training of GCWC and A-GCWC is $O((F^2K + n^2F) \times m \times S)$ and $O((F^2K+n^2F+nKF)\times m\times S)$, respectively, where F is the maximum number of convolutional filters, K is the maximum size of the convolutional filters, n is the number of edges, S is the batch size, and m is the number of histogram buckets. The average running time on CI for training is 36 ms (16 ms) per training batch for A-GCWC (GCWC), where each training batch has 20 training instances, i.e., 20 input matrices. The average running time on HW for training is 19 ms (14 ms)

Type	Data	Model	Configuration	#Para	LR	Decay	Dropout	Regul
HIST	HW	CNN	$C_{16}^{8\times1}$-P_4-$C_{16}^{8\times1}$-P_2-FC_{24}	18,840	3.5e-3	0.95	0.3	0.001
		DR	$C_{16}^{8\times1}$-$C_{16}^{8\times1}$-$C_{16}^{8\times1}$-$C_{8}^{8\times1}$	39,680	6.4e-3	0.92	0.0	0.004
		GCWC	$C_{16}^{8\times1}$-P_4-$C_{16}^{8\times1}$-P_2-FC_{24}	19,224	2.0e-4	0.999	0.17	0.001
		A-GCWC	$C_{16}^{8\times1}$-P_4-$C_{16}^{8\times1}$-P_2-FC_{24} + $C_{4}^{2\times2}$-P_2-$C_{8}^{2\times2}$-P_2-FC_1	20,184	5.0e-4	0.98	0.2	0.045
	CI	CNN	$C_{8}^{8\times1}$-P_2-$C_{8}^{4\times1}$-P_2-FC_{172}	32,412	2.2e-3	0.97	0.12	0.001
		DR	$C_{16}^{8\times1}$-$C_{16}^{8\times1}$-$C_{16}^{8\times1}$-$C_{8}^{8\times1}$	39,680	0.01	1.0	0.6	0.1
		GCWC	$C_{8}^{8\times1}$-P_2-$C_{8}^{4\times1}$-P_2-FC_{172}	46,860	1.5e-3	0.99	0.13	0.002
		A-GCWC	$C_{8}^{8\times1}$-P_2-$C_{8}^{4\times1}$-P_2-FC_{172} + $C_{4}^{2\times2}$-P_2-$C_{8}^{2\times2}$-P_2-FC_1	48,296	1.0e-3	0.99	0.19	0.004
AVG	HW	CNN	$C_{16}^{8\times1}$-P_4-$C_{16}^{8\times1}$-P_2-FC_{24}	3,384	3.5e-3	0.95	0.3	0.001
		DR	$C_{16}^{8\times1}$-$C_{16}^{8\times1}$-$C_{16}^{8\times1}$-$C_{1}^{8\times1}$	33,520	0.1	0.9	0.0	0.1
		GCWC	$C_{16}^{8\times1}$-P_4-$C_{16}^{8\times1}$-P_2-FC_{24}	3,768	2.0e-4	0.999	0.17	0.001
		A-GCWC	$C_{16}^{8\times1}$-P_4-$C_{16}^{8\times1}$-P_2-FC_{24} + $C_{4}^{2\times2}$-P_2-$C_{8}^{2\times2}$-P_2-FC_1	4,728	5.0e-4	0.98	0.2	0.045
	CI	CNN	$C_{8}^{8\times1}$-P_2-$C_{8}^{4\times1}$-P_2-FC_{172}	30,088	2.2e-3	0.97	0.12	0.001
		DR	$C_{16}^{8\times1}$-$C_{16}^{8\times1}$-$C_{16}^{8\times1}$-$C_{1}^{8\times1}$	33,520	0.013	0.9	0.6	0.1
		GCWC	$C_{8}^{8\times1}$-P_2-$C_{8}^{4\times1}$-P_2-FC_{172}	44,536	1.5e-3	0.99	0.13	0.002
		A-GCWC	$C_{8}^{8\times1}$-P_2-$C_{8}^{4\times1}$-P_2-FC_{172} + $C_{4}^{2\times2}$-P_2-$C_{8}^{2\times2}$-P_2-FC_1	45,972	1.0e-3	0.99	0.19	0.004

Table III: Model Construction and Hyperparamter Selection.

per training batch for A-GCWC (GCWC), where each training batch has 20 training instances.

5) Baselines: We compare GCWC and A-GCWC with five baselines: (1) Historical Average (HA): for each edge, we use all the travel speed records on the edge in the training data to derive a histogram. The histogram is used as the estimated histogram in the testing phase. (2) GP: a Gaussian process regression model. (3) RF: a random forest regression model. (4) LSM [9]: the state-of-the-art for filling in missing weights in road networks using a latent space model. Note that GP, RF, and LSM are originally only able to fill in deterministic weights. To support filling in stochastic weights, we fill in values for each bucket separately. For example, in the setting of HIST-8, we consider 8 individual regression problems, where each regression is able to fill in values for each bucket. (5) CNN: classical convolutional neural network, whose hyperparameters are set by following the same notations as for GCWC and A-GCWC, as shown in Table III. (6) DR: diffusion convolutional recurrent neural network [19], which is the state-of-the-art for predicting deterministic edge weights based on historical data.

6) Evaluation Metrics: We describe the metrics for evaluating the *estimation*, *prediction*, and *average* functions.

Estimation and Prediction: To assess the effectiveness of the models for estimation and prediction, we use the Mean Kullback-Leibler divergence Ratio (MKLR) and the Fraction of Likelihood Ratio (FLR) to measure the accuracy of the estimated (or predicted) stochastic weights $\widehat{W}$.

Specifically, MKLR is defined as follows.

$$MKLR = \frac{\sum_{i=1}^{T}\sum_{j=1}^{n} I_{ij} KL(w_{G_{j\cdot}^{(i)}} || \widehat{w}_{j\cdot}^{(i)})}{\sum_{i=1}^{T}\sum_{j=1}^{n} I_{ij} KL(w_{G_{j\cdot}^{(i)}} || HA_{j\cdot})}, \tag{11}$$

where T is the total number of testing time intervals, n is the total number of edges, and $I_{ij} \in [0,1]$, with $i \in [1,T]$ and $j \in [1,n]$, is an indicator of whether the stochastic weight of the j-th edge at the i-th time interval needs to be evaluated. In particular, we set set $I_{ij} = 0$, if in the i-th interval, edge e_j is not covered by traffic data; otherwise, we set $I_{ij} = 1$. Next $w_{G_{j\cdot}^{(i)}}$ and $\widehat{w}_{j\cdot}^{(i)}$ are the ground truth and estimated (or predicted) stochastic weights for the j-th edge at the i-th time interval, respectively.

Function $KL(\cdot||\cdot)$ computes the KL-divergence between two distributions, i.e., two stochastic weights represented as histograms. The lower a KL-divergence value is, the more similar the two stochastic weights are, indicating more accurate estimation or prediction. However, since KL-divergences range from 0 to ∞, it is hard to judge how small a KL-divergence value must be for an estimation or prediction to be considered accurate.

To solve this problem, we use $HA_{j\cdot}$ as a reference stochastic weight for the j-th edge, which is derived from all speed records in the training data that traversed the j-th edge, i.e., using the HA baseline. Here, we interpret $HA_{j\cdot}$ as the worst estimation or prediction of the stochastic weight for the j-th edge. Next, we derive a ratio $MKLR$ between the KL-divergence of another method and the KL-divergence of HA. This ratio suggests how much a method can improve over HA. Lower $MKLR$ values indicate higher improvements over HA.

We also measure the accuracy of estimated or predicted stochastic weights using FLR, defined as follows.

$$FLR = \frac{\sum_{i=1}^{T}\sum_{j=1}^{n} I_{ij} |LR_{ij} > 1|}{\sum_{i=1}^{T}\sum_{j=1}^{n} I_{ij}}, \tag{12}$$

where the meanings of T, n, and I_{ij} are the same as those in Equation (11), and LR_{ij} denotes the *likelihood ratio* [29] on the j-th edge in the i-th testing time interval. Specifically, $LR_{ij} = \frac{\sum_{k=1}^{N_{ij}} log(P_{\hat{w}}(o_k)+\epsilon)}{\sum_{k=1}^{N_{ij}} log(P_{HA}(o_k)+\epsilon)}$, where N_{ij} is the total number of ground truth speed records on the j-th edge in the i-th test time interval, o_k is the k-th ground truth speed record, and ϵ is a small value introduced to avoid zeros in *log* functions.

Given the j-th edge in the i-th testing time interval, we have an estimated or predicted stochastic weight $\hat{w}_{j\cdot}^{(i)}$ and the reference stochastic weight $HA_{j\cdot}$. We compute $P_{\hat{w}(o_k)}$ and $P_{HA}(o_k)$ as the likelihoods of observing o_k from the two distributions, i.e., $\hat{w}_{j\cdot}^{(i)}$ and $HA_{j\cdot}$. Here, if $LR_{ij} > 1$, the estimated (or predicted) weight $\hat{w}_{j\cdot}^{(i)}$ has a higher likelihood of observing the ground speed records than the reference weight

$HA_{j\cdot}$, thus indicating a more accurate prediction of $\hat{w}_{j\cdot}^{(i)}$ than when using $HA_{j\cdot}$. We set $|LR_{ij} > 1|$ as 1 if the LR_{ij} value exceeds 1; otherwise, it is set to 0.

Average: We use Mean Absolute Percentage Error (MAPE) to measure the accuracy of the estimated average speeds.

$$MAPE = \frac{\sum_{i=1}^{T}\sum_{j=1}^{n} I_{ij}\frac{|y_{ij}-\hat{y}_{ij}|}{y_{ij}}}{\sum_{i=1}^{T}\sum_{j=1}^{n} I_{ij}} \times 100\%, \quad (13)$$

where T, n, and I_{ij} have the same meaning as before. Further, y_{ij} and $\hat{y}_{ij}$ represent the ground truth and estimated average speeds for the j-th edge in the i-th time interval, respectively.

B. Experimental Results

We proceed to cover experimental results using data sets CI and HW with different removal ratios rm. We compare our models for *estimation* and *prediction* with baseline approaches and report MKLR and FLR values for both data sets. We also compare our models for the *average* function with the baseline approaches and report MAPE values. Finally, we report on a study of the scalability of the proposed models.

1) Estimation: Tables IV and V show MKLR values on HW and CI, respectively. As lower MKLR values indicate higher accuracy, we highlight the least MKLR values in each rm setting. A-GCWC has the best accuracy under all settings.

rm	GP	RF	LSM	CNN	DR	GCWC	A-GCWC
0.5	1.00	0.91	1.54	0.45	0.48	0.43	**0.43**
0.6	1.00	0.93	1.57	0.47	0.52	0.44	**0.43**
0.7	1.00	0.95	1.58	0.49	0.52	0.45	**0.44**
0.8	1.00	0.98	1.61	0.56	0.57	0.52	**0.46**

Table IV: MKLR for the HW Dataset, Estimation.

rm	GP	RF	LSM	CNN	DR	GCWC	A-GCWC
0.5	1.00	0.96	1.08	0.55	0.85	**0.48**	**0.48**
0.6	1.00	0.97	1.17	0.59	0.68	0.50	**0.49**
0.7	1.00	0.98	1.26	0.58	0.55	0.50	**0.49**
0.8	1.00	0.99	1.35	0.66	0.61	**0.49**	**0.49**

Table V: MKLR for the CI Dataset, Estimation.

For both data sets, the MKLR values of all methods increase as the removal ratio rm increase. The reason is that as the rm value increase, fewer edges are covered with traffic data, and therefore more edges need to be assigned estimated stochastic weights, which is more challenging. We observe that LSM, the state-of-the-art method in weight completion, fails in the considered settings, since all MKLR values exceed 1, meaning that HA is better than LSM. This suggest that LSM cannot be extended to support stochastic weights and LSM cannot deal with the case where many edges lack traffic data.

The MKLR values reported for CNN vary significantly as rm increases, while those reported by GCWC and A-GCWC, especially A-GCWC, are rather stable. This is because CNN is unable to capture the spatial correlations of a road network well, while GCWC and A-GCWC are.

DR achieves much better accuracy on HW than on CI—in particular, DR has MKLR as large as 0.85 at $rm = 0.5$ on CI. This indicates that although DR has good propagation abilities on small graphs, this ability is weakened on large graphs.

Finally, GCWC and A-GCWC show stable accuracy, and they double the accuracy of HA, i.e., the MKLR values of below 0.5 for both data sets. A-GCWC reports more stable MKLR values when rm values increase and higher accuracy, i.e., lower MKLR values, than GCWC.

Next, we report FLR values for each rm setting, as show in Tables VI and VII. We highlight the highest FLR values for each rm setting since higher FLR values indicate higher accuracy.

For both data sets, we observe that the FLR values of all methods decrease as rm increases, which is because the tasks become more challenging. LSM also fails and reports FLR values less than 0.5, meaning that HA behaves better than LSM in most cases. In most settings, GCWC and A-GCWC can achieve the highest FLR values, and A-GCWC outperforms GCWC, which is consistent with the results reported by the MKLR values. CNN slightly outperforms our models given $rm = 0.5$ in HW data set. This is because for a small road network, e.g., for HW, and when many edges are covered with traffic data, e.g., $rm = 0.5$, CNN may be able to capture some latent correlations among the stochastic weights of edges. DR shows consistent trends—it achieves good accuracy on HW and performs significantly worse on CI.Overall, A-GCWC achieves the best FLR.

rm	GP	RF	LSM	CNN	DR	GCWC	A-GCWC
0.5	0.52	0.78	0.21	**0.90**	0.86	0.88	0.89
0.6	0.50	0.76	0.20	0.88	0.88	0.88	**0.92**
0.7	0.48	0.74	0.20	0.88	0.89	0.89	**0.91**
0.8	0.47	0.67	0.22	0.85	0.87	0.84	**0.88**

Table VI: FLR for the HW Dataset, Estimation.

rm	GP	RF	LSM	CNN	DR	GCWC	A-GCWC
0.5	0.52	0.61	0.10	0.78	0.75	0.84	**0.85**
0.6	0.52	0.61	0.11	0.78	0.75	0.83	**0.84**
0.7	0.51	0.60	0.10	0.81	0.81	0.83	**0.84**
0.8	0.52	0.60	0.11	0.77	0.78	**0.85**	0.83

Table VII: FLR for the CI Dataset, Estimation.

2) Prediction: Tables VIII and IX show MKLR values for HW and CI, respectively. We highlight the least MKLR values given each rm setting.

Most observations for *estimation* also hold for *prediction*. On HW, as rm increases, the MKLR values of all methods, except GP and RF, follow an increasing trend. In contrast, on CI, the MKLR values do not follow an increase trend when rm increases. This is because CI is a larger, city network with more uncertainty and less dependency among different time intervals than in the case of HW, which is a highway toll gate network. Nevertheless, GCWC and A-GCWC outperform the other methods in almost all settings for both data sets, and A-GCWC behaves more stable and has lower MKLR values than does GCWC.

Next, we report FLR in Tables X and XI. In this setting, we use real speed observations during the testing intervals to

rm	GP	RF	LSM	CNN	DR	GCWC	A-GCWC
0.5	2.61	1.00	1.56	0.45	0.47	0.46	**0.43**
0.6	2.59	1.00	1.60	0.45	0.46	0.46	**0.44**
0.7	2.52	0.99	1.62	0.46	0.46	0.47	**0.43**
0.8	2.60	0.98	1.67	0.46	0.49	0.47	**0.45**

Table VIII: MKLR for the HW Dataset, Prediction.

rm	GP	RF	LSM	CNN	DR	GCWC	A-GCWC
0.5	1.09	0.97	4.18	0.50	0.48	0.45	**0.43**
0.6	1.12	0.97	4.15	0.50	0.49	**0.46**	**0.46**
0.7	1.16	0.98	4.16	0.54	0.54	0.49	**0.48**
0.8	1.24	0.98	4.30	0.59	0.53	0.50	**0.49**

Table IX: MKLR for the CI Dataset, Prediction.

compute the likelihood ratios. Recall that higher FLR values indicate higher accuracy.

rm	GP	RF	LSM	CNN	DR	GCWC	A-GCWC
0.5	0.29	0.55	0.23	0.93	0.93	**0.96**	0.92
0.6	0.31	0.55	0.22	0.93	0.93	**0.96**	0.92
0.7	0.32	0.58	0.23	0.93	**0.94**	0.93	0.93
0.8	0.35	0.59	0.24	0.93	**0.93**	**0.93**	0.92

Table X: FLR for the HW Dataset, Prediction.

rm	GP	RF	LSM	CNN	DR	GCWC	A-GCWC
0.5	0.50	0.58	0.12	0.84	0.82	**0.86**	0.85
0.6	0.49	0.58	0.12	0.83	0.83	0.84	**0.85**
0.7	0.49	0.58	0.13	0.84	0.82	**0.84**	**0.84**
0.8	0.47	0.57	0.14	0.82	0.82	**0.84**	0.83

Table XI: FLR for the CI Dataset, Prediction.

We observe that LSM also fails, with all values being below 0.5. However, we cannot observe a clear trend with the increase of rm for both data sets. Further, we observe that GCWC and A-GCWC achieve the best results in most settings, the exception being when $rm = 0.7$ (Table X). Overall, the performance improvements of GCWC and A-GCWC over DR are less obvious for prediction, because DR models temporal dependency explicitly by using RNNs, yielding more accurate predictions the edges with data. However, the overall accuracies of GCWC and A-GCWC are still better due to ability of these to propagate weights to edges without data.

We also observe that the FLR values for CNN, DR, GCWC, and A-GCWC are closer and higher for HW than those for CI, because speed observations for CI have more uncertainty than for HW, making predictions for CI more challenging.

3) Average: Tables XII and XIII show MAPE when estimating average speeds for CI and HW with different rm values, respectively. In this setting, LSM is the state-of-the-art linear solution [9] and DR [19] can be regarded as the state-of-the-art non-linear solution.

We have the following observations. (1) A-GCWC performs overall best on both data sets; (2) LSM seems to not work on the CI dataset. The reason may be that the citywide road network is much more complex than the highway road network, meaning that linear modeling is unable to capture the latent attributes of this system. On the HW dataset, LSM does a much better job when the rm value is 0.5, compared to on the CI dataset. However, the performance falls markedly when the rm value increases, meaning that LSM falls short when many edges are not covered by traffic data. Note that in the original paper [9], LSM shows good accuracy when up to 30% of the edges lack average speeds, i.e., when $rm \leq 0.3$. (3) CNN and DR also performs better than LSM, suggesting that it the linear modeling correlations is the key problem. (4) While DR is the state-of-the-art when the data is not sparse, it performs worse than the proposed GCWC and A-GCWC when the data is sparse.

rm	LSM	CNN	DR	GCWC	A-GCWC
0.5	25.5%	12.9%	14.5%	13.0%	**12.8%**
0.6	28.5%	13.0%	13.5%	13.0%	**12.9%**
0.7	33.5%	13.0%	13.3%	**12.1%**	12.9%
0.8	48.3%	13.0%	13.5%	**12.4%**	12.9%

Table XII: MAPE for the HW Dataset, Average.

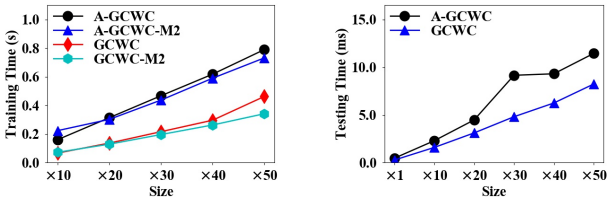
rm	LSM	CNN	DR	GCWC	A-GCWC
0.5	31.0%	10.9%	11.3%	11.6%	**10.8%**
0.6	37.3%	**11.2%**	12.5%	12.2%	**11.2%**
0.7	44.7%	11.5%	13.6%	12.2%	**11.4%**
0.8	52.1%	13.0%	**11.5%**	12.1%	**11.5%**

Table XIII: MAPE for the CI Dataset, Average.

4) Scalability: We conduct this experiment to investigate the scalability of GCWC and A-GCWC on large road networks. Due to the unavailability of large road networks with sufficient amount of traffic data that cover most edges during most intervals, we manually enlarge the road network of CI by scales of 10, 20, 30, 40, and 50 such that the largest road network has a total number of $172 \times 50 = 8,600$ edges. Of course, the road network can be enlarged further, e.g., 60 or higher. However, we found that this is the largest road network that one single K80 GPU can deal with in our setting.

If the road network is too large to fit into a single machine, we can divide the network into smaller sub-networks and process them either in parallel on multiple computers [30], [31] or in sequence on a single computer. To simulate the case of a very large road network, we consider two settings: (1) processing a single road network using GCWC and A-GCWC; and (2) partitioning the road network into two small road networks and processing the two small road networks in sequence, denoted as GCWC-M2 and A-GCWC-M2.

Figure 6(a) shows the average training time for one batch with batch size 20. This is the time it takes to finish an entire back propagation using 20 training instances, i.e., 20 input matrices. We observe that training A-GCWC takes more time than GCWC, which is reasonable since A-GCWC needs to train an extra CP-CNN. Further, if we partition a large road network into two small sub-networks and train them in sequence, this takes less time. However, this normally results in lower accuracy since the partitioning destroys some adjacency relationships in the original road network.



(a) Training Time (b) Testing Time

Figure 6: Scalability.

Figure 6(b) shows that the average testing time for one instance, e.g., estimating or predicting a weight matrix $\widehat{W}$, is very fast (less than 15 ms), and there is little difference between A-GCWC and GCWC.

VII. Conclusions and Outlook

High-resolution vehicle routing calls for road network models where each edge has a time-varying travel-time distribution. Even with massive volumes of vehicle data, it is generally impossible to construct distributions, also called stochastic weights, for all edges and times. We define and study the problem of stochastic weight completion in this setting. A data-driven deep learning model, graph convolutional weight completion (GCWC), is proposed to fill in missing stochastic weights. In addition, an advanced GCWC model that takes into account contextual information is proposed to further improve accuracy. Empirical studies with two different, real datasets—highway loop detector data and citywide taxi GPS data—suggest that the proposed models are capable of outperforming other methods in all the experimental settings considered.

In future work, it is of interest to exploit temporal correlations to further improving the accuracy of the stochastic weights and to support continuous distribution models such as Gaussian mixture models. It is also of interest to integrate GCWC with existing routing algorithms [32], [33] to enhance routing quality.

Acknowledgment

This research was supported in part by a grant from the Obel Family Foundation, a grant from the Independent Research Fund Denmark, and by the DiCyPS center funded by Innovation Fund Denmark.

References

[1] S. Aljubayrin, B. Yang, C. S. Jensen, and R. Zhang, "Finding non-dominated paths in uncertain road networks," in *SIGSPATIAL*, pp. 15:1–15:10, 2016.

[2] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang, "Stochastic skyline route planning under time-varying uncertainty," in *ICDE*, pp. 136–147, 2014.

[3] G. Andonov and B. Yang, "Stochastic shortest path finding in path-centric uncertain road networks," in *MDM*, pp. 40–45, 2018.

[4] J. Hu, B. Yang, C. Guo, and C. S. Jensen, "Risk-aware path selection with time-varying, uncertain travel costs: a time series approach," *VLDB J.*, vol. 27, no. 2, pp. 179–200, 2018.

[5] J. Hu, B. Yang, C. S. Jensen, and Y. Ma, "Enabling time-dependent uncertain eco-weights for road networks," *GeoInformatica*, vol. 21, no. 1, pp. 57–88, 2017.

[6] B. Yang, J. Dai, C. Guo, C. S. Jensen, and J. Hu, "PACE: a path-centric paradigm for stochastic path finding," *VLDB J.*, vol. 27, no. 2, pp. 153–178, 2018.

[7] C. Guo, C. S. Jensen, and B. Yang, "Towards total traffic awareness," *SIGMOD Record*, vol. 43, no. 3, pp. 18–23, 2014.

[8] Z. Ding, B. Yang, Y. Chi, and L. Guo, "Enabling smart transportation systems: A parallel spatio-temporal database approach," *IEEE Trans. Computers*, vol. 65, no. 5, pp. 1377–1391, 2016.

[9] D. Deng, C. Shahabi, U. Demiryurek, L. Zhu, R. Yu, and Y. Liu, "Latent space model for road networks to predict time-varying traffic," in *SIGKDD*, pp. 1525–1534, 2016.

[10] J. Dai, B. Yang, C. Guo, C. S. Jensen, and J. Hu, "Path cost distribution estimation using trajectory data," *PVLDB*, vol. 10, no. 3, pp. 85–96, 2016.

[11] C. Guo, B. Yang, J. Hu, and C. S. Jensen, "Learning to route with sparse trajectory sets," in *ICDE*, pp. 1073–1084, 2018.

[12] T. Idé and M. Sugiyama, "Trajectory regression on road networks," in *AAAI*, pp. 203–208, 2011.

[13] J. Zheng and L. M. Ni, "Time-dependent trajectory regression on road networks via multi-task learning," in *AAAI*, pp. 1048–1055, 2013.

[14] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering," *arXiv:1606.09375 [cs, stat]*, 2016. arXiv: 1606.09375.

[15] B. Yang, C. Guo, and C. S. Jensen, "Travel cost inference from sparse, spatio-temporally correlated time series using markov models," *PVLDB*, vol. 6, no. 9, pp. 769–780, 2013.

[16] Z. Ding, B. Yang, R. H. Güting, and Y. Li, "Network-matched trajectory-based moving-object database: Models and applications," *IEEE Trans. Intelligent Transportation Systems*, vol. 16, no. 4, pp. 1918–1928, 2015.

[17] 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, 2014.

[18] R. Yu, Y. Li, C. Shahabi, U. Demiryurek, and Y. Liu, "Deep learning: A generic approach for extreme condition traffic forecasting," in *SIAMICDM*, pp. 777–785, 2017.

[19] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *ICLR*, 2018.

[20] R. Cirstea, D. Micu, G. Muresan, C. Guo, and B. Yang, "Correlated time series forecasting using multi-task deep neural networks," in *CIKM*, 2018.

[21] Y. Li, K. Fu, Z. Wang, C. Shahabi, J. Ye, and Y. Liu, "Multi-task representation learning for travel time estimation," in *KDD '18*, pp. 1695–1704, 2018.

[22] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Distinguishing trajectories from different drivers using incompletely labeled trajectories," in *CIKM*, 2018.

[23] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Unsupervised and Transfer Learning@ICML*, pp. 37–50, 2012.

[24] T. Kieu, B. Yang, and C. S. Jensen, "Outlier detection for multidimensional time series using deep neural networks," in *MDM*, pp. 125–134, 2018.

[25] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral Networks and Locally Connected Networks on Graphs," *arXiv:1312.6203*, 2013.

[26] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," *arXiv:1609.02907*, Sept. 2016.

[27] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, pp. 3111–3119, 2013.

[28] "Tensor flow api." https://www.tensorflow.org/api_docs/python/tf/nn/embedding_lookup.

[29] R. Waury, J. Hu, B. Yang, and C. S. Jensen, "Assessing the accuracy benefits of on-the-fly trajectory selection in fine-grained travel-time estimation," in *IEEE MDM*, pp. 240–245, 2017.

[30] B. Yang, Q. Ma, W. Qian, and A. Zhou, "TRUSTER: trajectory data processing on clusters," in *DASFAA*, pp. 768–771, 2009.

[31] P. Yuan, C. Sha, X. Wang, B. Yang, A. Zhou, and S. Yang, "XML structural similarity search using mapreduce," in *WAIM*, pp. 169–181, 2010.

[32] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k shortest paths with diversity," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 3, pp. 488–502, 2018.

[33] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k optimal sequenced routes," in *ICDE*, pp. 569–580, 2018.