AutoSTG: Neural Architecture Search for Predictions of Spatio-Temporal Graphs*

Zheyi Pan[1,2,3], Songyu Ke[1,2,3], Xiaodu Yang[4], Yuxuan Liang[6]
Yong Yu[1], Junbo Zhang[2,3,4], Yu Zheng[2,3,4,5]
[1]Department of Computer Science and Engineering, Shanghai Jiaotong University, China
[2]JD iCity & [3]JD Intelligent Cities Research, JD Tech, Beijing, China
[4]Artificial Intelligence Institute, Southwest Jiaotong University, China
[5]School of Computer Science and Technology, Xidian University, China
[6]School of Computing, National University of Singapore, Singapore
{zheyi.pan,songyu-ke,yuxliang,msjunbozhang,msyuzheng}@outlook.com;
xiaodu.yang@foxmail.com;yyu@apex.sjtu.edu.cn

ABSTRACT

Spatio-temporal graphs are important structures to describe urban sensory data, *e.g.,* traffic speed and air quality. Predicting over spatio-temporal graphs enables many essential applications in intelligent cities, such as traffic management and environment analysis. Recently, many deep learning models have been proposed for spatio-temporal graph prediction and achieved significant results. However, designing neural networks requires rich domain knowledge and expert efforts. To this end, we study *automated* neural architecture search for spatio-temporal graphs with the application to urban traffic prediction, which meets two challenges: 1) how to define search space for capturing complex spatio-temporal correlations; and 2) how to learn network weight parameters related to the corresponding attributed graph of a spatio-temporal graph.

To tackle these challenges, we propose a novel framework, entitled AutoSTG, for automated spatio-temporal graph prediction. In our AutoSTG, spatial graph convolution and temporal convolution operations are adopted in our search space to capture complex spatio-temporal correlations. Besides, we employ the meta learning technique to learn the adjacency matrices of spatial graph convolution layers and kernels of temporal convolution layers from the meta knowledge of the attributed graph. And specifically, such meta knowledge is learned by a graph meta knowledge learner that iteratively aggregates knowledge on the attributed graph. Finally, extensive experiments were conducted on two real-world benchmark datasets to demonstrate that AutoSTG can find effective network architectures and achieve state-of-the-art results. To the best of our knowledge, we are the first to study neural architecture search for spatio-temporal graphs.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Information systems** → **Spatial-temporal systems**.

*Junbo Zhang, Yu Zheng and Yong Yu are corresponding authors.

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449816

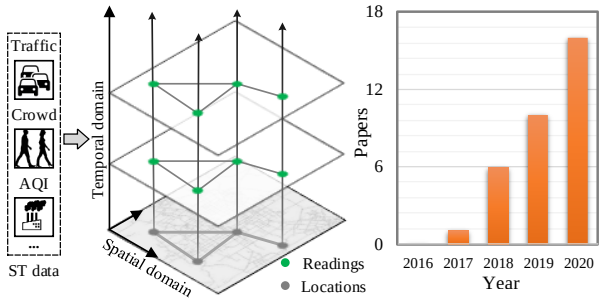
KEYWORDS

Neural architecture search, spatio-temporal graph, meta learning

ACM Reference Format:
Zheyi Pan[1,2,3], Songyu Ke[1,2,3], Xiaodu Yang[4], Yuxuan Liang[6] and Yong Yu[1], Junbo Zhang[2,3,4], Yu Zheng[2,3,4,5]. 2021. AutoSTG: Neural Architecture Search for Predictions of Spatio-Temporal Graphs. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3442381.3449816

1 INTRODUCTION

Recent advances of data acquisition technology help collect a variety of spatio-temporal (ST) data in urban areas, such as urban traffic, air quality, and etc. Such data has complex spatial and temporal correlations [25, 26], which can be depicted by spatio-temporal graphs (STG), as shown in Figure 1(a). Hence, predicting STGs can enable many mission-critical applications, such as traffic management and environment analysis [30], which are essential to the development of intelligent cities.



(a) ST data and the related ST graph structure in urban areas

(b) Numbers of papers to model ST graph in the recent years

Figure 1: Background of spatio-temporal graph prediction.

Recently, a growing number of ST neural networks have been proposed for STG prediction, as our statistics of related papers (published in the top data mining and AI conferences) show in Figure 1(b). To capture the spatial and temporal correlations, many representative models, such as [12, 13, 21, 23], are carefully assembled from small network structures, which can be grouped into temporal networks (*e.g.,* convolution neural networks, recurrent neural networks, and temporal attention networks) and spatial networks (*e.g.,* graph convolution networks and spatial attention

networks). By leveraging the capability of modeling ST correlations, these models can achieve significant performance.

However, these representative models are subjected to specific data types or data distributions, and directly transferring these networks to other tasks could reduce prediction accuracy. For example, in traffic prediction, the data of different cities can show heterogeneous spatial correlations. Some cities (*e.g.,* Beijing and Shanghai) meet severe traffic problems, where traffic jams are often intensified and broadcast to a large area, showing long-distance spatial correlations. Whereas in some other cities, there is much less traffic congestion, which hardly impacts moving vehicles, showing less significance of long-distance spatial correlations. Because of such disparity, it is necessary to design neural networks for different tasks, which requires substantial domain knowledge and large amounts of expert efforts, making it time-consuming and costly to popularize deep learning technology to a variety of important applications. Therefore, there is a rising demand for an automated neural architecture search (NAS) framework for STG prediction.

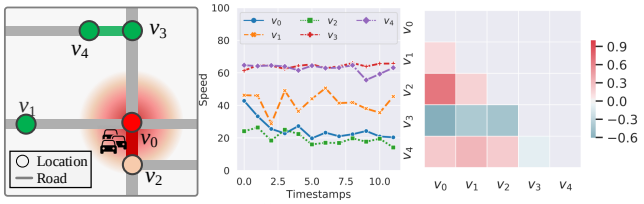
In the beginning, NAS frameworks were proposed for conventional image recognition and sequence modeling [15, 18, 22]. Thereafter, these techniques enabled other applications, such as modeling graph data [5, 31] and predicting grid-based ST data [11]. However, these existing methods cannot be directly adopted to model STGs, because of the following two problems.

Problem 1: *at the network architecture level, how to define the search space for modeling STGs?* The basic assumption of STGs is that the readings of a node are conditioned on its historical readings, as well as the readings of its neighborhoods [21]. The key problem is to capture the spatial and temporal correlations in such a non-Euclidean and irregular space. This means that the search space for the conventional data types, including sequence data, grid-based data, and graph data, cannot be directly adopted in STG prediction.

Problem 2: *at the network parameter level, how to learn the weight parameters related to the attributed graph of an STG?* In general, an STG is associated with an attributed graph, which impacts ST correlations. Figure 2(a) presents a real-world traffic example in METR-LA dataset [12], that speed sensors in a city form an attributed graph, where nodes and edges denote sensors and relations between sensors, respectively. These nodes and edges have some geographical attributes, *e.g.,* node locations and road distance between nodes. The characteristics of this graph, *i.e.,* the characteristics of these nodes and edges, indicate the ability of the transportation system, impacting how traffic broadcasts in the STG [16].

As network weight parameters represent to what extent the data is correlated in an STG, it is crucial to capture the relationships between network weight parameters and the characteristics of the attributed graph. However, such characteristics are complex, depending on both the attributes and graph structure:

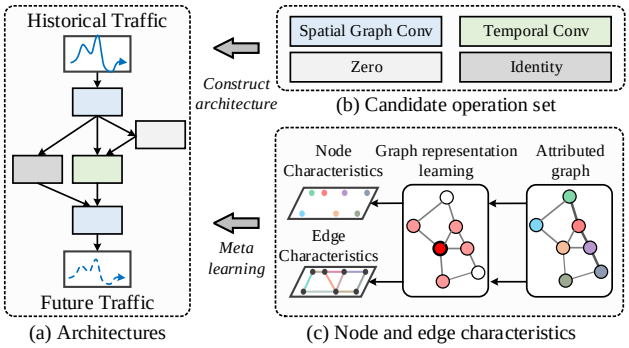
- The characteristics of nodes and edges are related to their own attributes. As the example shows in Figure 2(a), v_0 is located at a hub while v_1 is an intermediate node on road network. Consequently, as shown in Figure 2(b), v_0 easily becomes congested, while v_1 is often unobstructed, showing diverse node characteristics. Similarly, as edge $\langle v_0, v_2\rangle$ is much shorter than edge $\langle v_0, v_1\rangle$, node v_0 has stronger impacts on v_2 than v_1 as depicted in Figure 2(c), showing diverse edge characteristics. Therefore, attributes can reflect the traffic-related characteristics.
- The characteristics of nodes and edges are related to the graph structure. First, characteristics of a node are related to its neighbors and edges. In Figure 2(a), v_2 and v_3 have very similar node attributes (*i.e.,* the density and structure of roads). However, as edge $\langle v_0, v_2\rangle$ is much shorter than $\langle v_0, v_3\rangle$, v_2 is more likely to witness traffic congestion than v_3, showing different node characteristics. Likewise, the characteristics of an edge are related to its two connecting nodes. In Figure 2(a), edge $\langle v_0, v_2\rangle$ and $\langle v_3, v_4\rangle$ have similar attributes (*i.e.,* length and width). However, v_0 and v_2 are easily blocked by traffic, while v_3 and v_4 are of easier traffic dispersion. Accordingly, in Figure 2(c), edge $\langle v_0, v_2\rangle$ shows much stronger correlations than $\langle v3, v4\rangle$, indicating different edge characteristics. Hence, it is essential to learning traffic-related characteristics taking into consideration the graph structure.



(a) Attributed graph (b) Traffic speed (mile/hour) (c) Edge correlations

Figure 2: Impacts of attributed graph on ST data.

To tackle all above problems, we propose a NAS framework, entitled AutoSTG, for spatio-temporal graph prediction. Without loss of generality, urban traffic prediction is employed as a concrete application in our study. To capture ST correlations, our search space is constructed from a candidate operation set, including *spatial graph convolution (SC)*, *temporal convolution (TC)*, and other widely used operations in NAS, *i.e., zero* and *identity*, as shown in Figure 3(a) and (b). As ST correlations are related to attributed graph, we employ meta learning [16] to generate weight parameters of SC and TC from node and edge meta knowledge (*i.e.,* characteristics) of attributed graph, as shown in Figure 3(a) and (c). Particularly, a graph representation learning network is used to learn the meta knowledge by aggregating neighbors' information on attributed graph, as shown in Figure 3(c). Our contributions are three-fold:



(a) Architectures (b) Candidate operation set (c) Node and edge characteristics

Figure 3: Insights of our NAS framework.

- We proposed a novel framework, entitled AutoSTG, for automated spatio-temporal graph prediction. Specifically, spatial graph convolution and temporal convolution operations are adopted

in our AutoSTG to capture spatial and temporal correlations, respectively. To the best of our knowledge, we are the first to study NAS framework for modeling STGs.
- To capture the ST correlations related to node and edge characteristics, meta learning technique is adopted to learn the adjacency matrices of spatial graph convolution layers and kernels of temporal convolution layers from the meta knowledge of the attributed graph. In particular, a graph meta knowledge learner, which is composed of node learners and edge learners, iteratively aggregates the neighbors' information of attributed graph to learn node and edge meta knowledge.
- We conduct extensive experiments on two widely used real-world benchmark datasets to verify our framework. The experimental results demonstrate that our AutoSTG can find effective neural architectures and achieve state-of-the-art prediction accuracy.

2 PRELIMINARIES

In this section, we first introduce the definitions and the problem statement. Then we describe the graph convolution layer used in our work. For brevity, frequently used notation is listed in Table 1.

Table 1: Frequently used notation.

Notation	Description
N_l, N_t	Number of locations/timestamps.
$N_{\text{in}}, N_{\text{out}}$	Timestamps of historical/future traffic.
$\mathbf{X}_t$	The traffic readings at timestamp t.
$\mathbf{V}^{(m)}$, $\mathcal{E}^{(m)}$	The node/edge representation at m-th iteration.
$\mathcal{A}$	The adjacency matrices of graph convolution.
$\mathbb{K}$	The kernels of temporal convolution.

2.1 Definitions and Problem Statement

Definition 1. ***Attributed graph.*** *An attributed graph is defined as* $\mathbb{G} = \left\{\mathbb{V}, \mathbb{E}, \mathbf{V}^{(0)}, \mathcal{E}^{(0)}\right\}$, *where* $\mathbb{V} = \left\{v_1, \cdots, v_{N_l}\right\}$ *denotes* N_l *nodes (locations),* $\mathbb{E} = \left\{\langle v_i, v_j\rangle \mid 1 \le i, j \le N_l\right\}$ *denotes edges (relationships between locations),* $\mathbf{V}^{(0)} \in \mathbb{R}^{N_l \times D}$ *denotes* D *dimensional attribute values for nodes, and* $\mathcal{E}^{(0)} \in \mathbb{R}^{N_l \times N_l \times K}$ *denotes* K *dimensional attribute values for edges, respectively. Note that if there is no edge between node* i *and* j *(i.e.,* $\langle v_i, v_j\rangle \notin \mathbb{E}$*), its attribute values in* $\mathcal{E}^{(0)}$ *are set as zero. In addition, we use* $\mathbb{N}_i = \left\{j \mid \langle v_i, v_j\rangle \in \mathbb{E}\right\}$ *to denote the neighbors of node* i.

Definition 2. ***STG prediction.*** *Given previous* N_{in} *readings for* N_l *nodes* $\left[\mathbf{X}_1, ..., \mathbf{X}_{N_{\text{in}}}\right] \in \mathbb{R}^{N_{\text{in}} \times N_l \times D}$ *and the attributed graph* $\mathbb{G}$, *predict the next* N_{out} *readings* $\left[\hat{\mathbf{Y}}_1, ..., \hat{\mathbf{Y}}_{N_{\text{out}}}\right] \in \mathbb{R}^{N_{\text{out}} \times N_l \times D}$.

Problem 1. ***NAS for STG prediction.*** *We aim to find a neural architecture for STG prediction, which learns from training dataset* $\mathbb{D}_{\text{train}}$ *and achieves the minimum loss on validation dataset* $\mathbb{D}_{\text{val}}$*:*

$$\begin{aligned} \min_{\Lambda} \quad & \mathrm{L}\left(\Theta^*(\Lambda), \Lambda, \mathbb{D}_{\text{val}}\right), \\ \text{s.t.} \quad & \Theta^*(\Lambda) = \arg\min_{\Theta} \mathrm{L}\left(\Theta, \Lambda, \mathbb{D}_{\text{train}}\right), \end{aligned} \tag{1}$$

where L, Θ, Λ *denote loss function, network weight parameters, and architecture parameters, respectively.*

2.2 Graph Convolution

Graph convolution [10] is capable of learning nodes' features given graph structure. Recently, [12] proposed diffusion convolution for modeling spatial correlations in traffic prediction. Thus, we employ diffusion convolution as the implementation of graph convolution.

Formally, given node state $\mathbf{H} \in \mathbb{R}^{N_l \times D}$ and K adjacency matrices $\mathcal{A} = [\mathbf{A}_1, \cdots, \mathbf{A}_K] = \left(a_{kij}\right)_{K \times N_l \times N_l}$ as inputs, diffusion convolution outputs node state $\mathrm{DC}(\mathbf{H}, \mathcal{A}, \mathbb{W}) \in \mathbb{R}^{N_l \times D'}$, computed by:

$$\mathrm{DC}\left(\mathbf{H}, \mathcal{A}, \mathbb{W}\right) = \sum_{k=1}^{K} \sum_{p=1}^{P} \left(\mathbf{A}_k\right)^p \mathbf{H}\mathbf{W}_{kp}, \tag{2}$$

where P is a constant denoting the number of diffusion steps, $(\mathbf{A}_k)^p$ is the power series of diffusion matrix $\mathbf{A}_k$, and $\mathbb{W} = \left\{\mathbf{W}_{kp} \in \mathbb{R}^{D \times D'}\right\}$ is the set of weight matrices for feature learning, respectively.

In STG prediction, the adjacency matrices $\mathcal{A}$ can be constructed from edge representation $\mathcal{E} = \left(e_{ijk}\right)_{N_l \times N_l \times K}$ by function $\mathrm{DM}(\mathcal{E}) = \left(a_{kij}\right)_{K \times N_l \times N_l}$. As the adjacency matrices represent the diffusion probability on the graph, we adopt softmax function to compute each a_{kij} by normalizing $\mathcal{E}$:

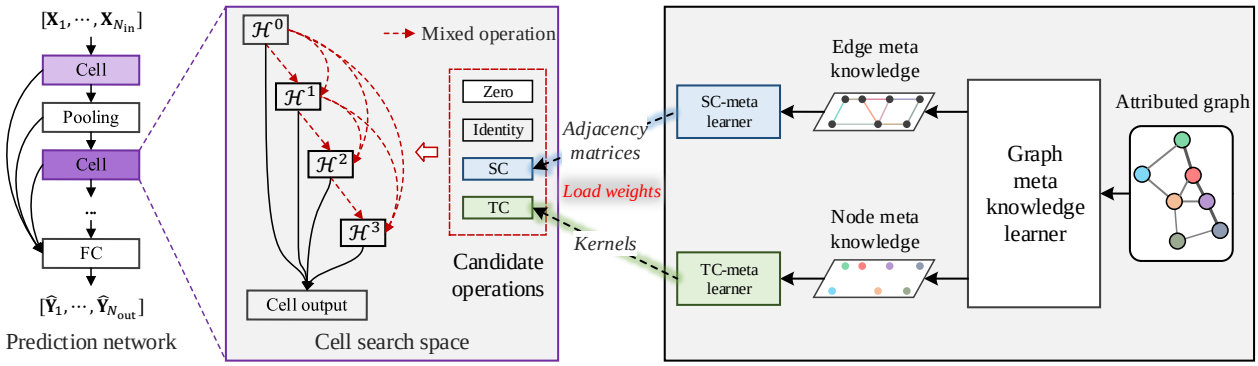
$$a_{kij} = \begin{cases} \frac{\exp(e_{ijk})}{\sum_{j' \in \mathbb{N}_i} \exp(e_{ij'k})}, & \langle v_i, v_j\rangle \in \mathbb{E}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

3 METHODOLOGIES

As shown in Figure 4, our framework consists of two parts: 1) constructing architectures from our search space, as shown in Figure 4(a); and 2) employing meta learning technique to learn the weights of SC and TC layers in the built architectures, as shown in Figure 4(b). Specifically in Figure 4(b), the meta learning part employs a graph meta knowledge learner to extract node and edge meta knowledge from the attributed graph, and then uses SC-meta learner and TC-meta learner to generate the adjacency matrices of SC layers and kernels of TC layers from the extracted meta knowledge, respectively, in the constructed architectures. In the following subsections, we first introduce the search space for architecture construction. Then we present the graph meta knowledge learner and meta learners. Finally, we show the optimization algorithm.

3.1 Search Space for Architecture Construction

Inspired by the existing convolutional-based neural architectures [21, 23] that have been proved to be effective in STG prediction, we design a convolutional-based search space for our framework. As shown in Figure 4(a), our prediction network is composed of a series of cells and temporal pooling layers, where the cells are employed for modeling ST correlations and the temporal pooling layers (*e.g.,* average pooling in temporal domain) are adopted to increase the temporal receptive fields of hidden states. All the outputs of cells and pooling layers are aggregated by shortcut connections for modeling multi-resolution ST correlations, and then a fully-connected (FC) layer is adopted to predict the next N_{out}-timestamp values. Initially, these cells have undetermined architectures, and our goal is to search the architecture for each of them.



(a) Architecture search space (b) Meta learning for network weight parameters

Figure 4: Framework Overview of AutoSTG.

Following DARTS framework [15], the search space of each cell is a direct acyclic graph, consisting of N_v vertices. Each vertex denotes a latent representation of the STG, *i.e.,* $\mathcal{H}^i \in \mathbb{R}^{T\times N_l\times D}$, where T is the number of timestamps and D is the number of features (T and D are fixed in each cell). The cell input is $\mathcal{H}^0$, while the output is the sum of all intermediate representations $\sum_{i=0}^{N_v}\mathcal{H}^i$. Each vertex pair (i,j) is associated with a candidate operation set $\mathbb{O}=\{o_1,o_2,\cdots\}$ (*i.e.,* a function set) that transforms $\mathcal{H}^i$, weighted by architecture parameters $\boldsymbol{\alpha}^{(i,j)}=\{\alpha_o^{(i,j)}\mid o\in\mathbb{O}\}$. The representation of a vertex can be computed based on all its predecessors:

$$\mathcal{H}^j=\sum_{i<j}\sum_{o\in\mathbb{O}}\frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o'\in\mathbb{O}}\exp\left(\alpha_{o'}^{(i,j)}\right)}o\left(\mathcal{H}^i\right). \tag{4}$$

For each pair (i,j), the operation with the highest score $\alpha_o^{(i,j)}$ in $\boldsymbol{\alpha}^{(i,j)}$ is selected as the operation in the final architecture.

To capture ST correlations, spatial graph convolution and temporal convolution are included in our candidate operation set $\mathbb{O}$:

- *Spatial Graph Convolution.* We use a diffusion convolution to capture spatial correlations of STGs. Formally, suppose that the input hidden state $\mathcal{H}=[\mathbf{H}_1,\cdots,\mathbf{H}_T]\in\mathbb{R}^{T\times N_l\times D}$, the input adjacency matrices is denoted as $\mathcal{A}$, and the learnable weights is denoted as $\mathbb{W}$. The spatial graph convolution function is defined as $\mathrm{SC}(\mathcal{H},\mathcal{A},\mathbb{W})=\mathcal{H}'=\left[\mathbf{H}'_1,\cdots,\mathbf{H}'_T\right]\in\mathbb{R}^{T\times N_l\times D}$, which employs diffusion convolution on each $\mathbf{H}_i$, expressed as:

$$\mathbf{H}'_i=\mathrm{DC}\left(\mathbf{H}_i,\mathcal{A},\mathbb{W}\right), \tag{5}$$

  where the computation of function $\mathrm{DC}(\cdot)$ and the constraints of the related parameters (*e.g.,* $\mathbb{W}$) refer to Equation (2).
- *Temporal Convolution.* We employ a temporal convolution on each node to capture temporal correlations of STGs. Suppose the input hidden state $\mathcal{H}\in\mathbb{R}^{T\times N_l\times D}$ is composed of $\left\{\mathbf{H}_1,\cdots,\mathbf{H}_{N_l}\right\}$, where each $\mathbf{H}_i\in\mathbb{R}^{T\times D}$ denotes the hidden state of node i. Then, given the input convolution kernels $\mathbb{K}=\left\{\mathcal{K}_1,\cdots,\mathcal{K}_{N_l}\right\}$, the temporal convolution function $\mathrm{TC}(\mathcal{H},\mathbb{K})=\mathcal{H}'$, where $\mathcal{H}'\in\mathbb{R}^{T\times N_l\times D}$ is composed of $\left\{\mathbf{H}'_1,\cdots,\mathbf{H}'_{N_l}\mid\mathbf{H}'_i\in\mathbb{R}^{T\times D}\right\}$. Then, the output hidden state of node i is computed by:

$$\mathbf{H}'_i=\mathbf{H}_i\star\mathcal{K}_i, \tag{6}$$

  where $\star$ denotes 1D convolution [9] for sequence modeling.

Besides, two widely used operations, *i.e., zero* and *identity* are also included in our candidate operation set like [15], to enable multiple branches and shortcut connections in network architectures.

3.2 Graph Meta Knowledge Learner

In STG prediction, the ST correlations of data are related to the characteristics of attributed graph $\mathbb{G}$. Therefore, it is essential to learn the representations, namely the meta knowledge, of nodes and edges from $\mathbb{G}$. Previously, [16] employed FC layers to learn node and edge meta knowledge respectively from node and edge attributes. However, such method ignores the impacts of graph structure. To tackle this problem, we propose a graph meta knowledge learner, which iteratively learns nodes' and edges' representations by aggregating their neighbors' information on $\mathbb{G}$.

As shown in Figure 5, we apply M iterations to learn node and edge representations by node learners and edge learners. Let $\mathbf{V}^{(m)}$ and $\mathcal{E}^{(m)}$ denote the node and edge representations at m-th iteration, respectively. The inputs are node and edge attributes, denoted as $\mathbf{V}^{(0)}$ and $\mathcal{E}^{(0)}$, and the outputs are node and edge meta knowledge, denoted as $\mathbf{V}^{(M)}$ and $\mathcal{E}^{(M)}$. The details of node learners and edge learners are as follow:

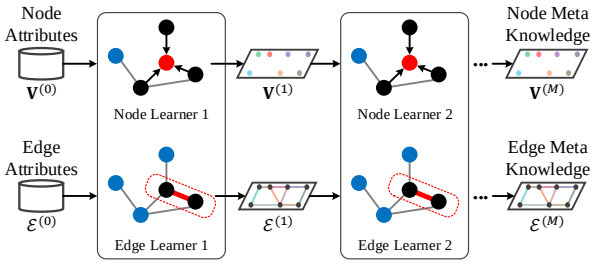


Figure 5: Structure of graph meta knowledge learner.

Node Learner. As characteristics of a node are related to its neighbors and connecting edges, we employ graph convolution to learn each node's representation by aggregating its neighbors' information through its edges. At m-th iteration, the node learner takes the representations of the previous iteration, *i.e.,* $\mathbf{V}^{(m-1)}\in\mathbb{R}^{N_l\times D}$ and $\mathcal{E}^{(m-1)}\in\mathbb{R}^{N_l\times N_l\times K}$ as inputs, and then returns the higher-level node representations $\mathbf{V}^{(m)}\in\mathbb{R}^{N_l\times D'}$. Without loss of generality, we employ diffusion convolution as a concrete example to show the implementation of node learner.

The first step is to compute adjacency matrices of graph convolution. In our work, we use edge representations to generate the adjacency matrices $\mathcal{A}$ by formula: $\mathcal{A} = \mathrm{DM}\left(\mathcal{E}^{(m-1)}\right)$. The computation of this function refers to Equation (3). Then the diffusion convolution can be employed for $\mathbf{V}^{(m-1)}$ to get the higher-level node representations by Equation (2), which can be expressed as:

$$\mathbf{V}^{(m)} = \mathrm{ReLU}\left(\mathrm{DC}\left(\mathbf{V}^{(m-1)}, \mathcal{A}, \mathbb{W}^{(m)}\right)\right), \tag{7}$$

where $\mathbb{W}^{(m)}$ denotes the learnable parameters of this node learner, and the constraint of $\mathbb{W}^{(m)}$ is illustrated in Section 2.2. In this way, we can learn the representations of each node by aggregating neighbors' information through the connected edges. Therefore, both the attributes and structure of attributed graph are considered in learning node representations.

Edge Learner. For each edge, its characteristics are related to its connected nodes. Thus, for each edge, we push the representations of its two connected nodes to this edge, and then use an FC layer to learn the higher-level edge representation. At m-th iteration, given node representations $\mathbf{V}^{(m-1)} = \left[\boldsymbol{v}_1^{(m-1)}, ..., \boldsymbol{v}_{N_l}^{(m-1)}\right] \in \mathbb{R}^{N_l \times D}$ and edge representations $\mathcal{E}^{(m-1)} \in \mathbb{R}^{N_l \times N_l \times K}$ as inputs, edge learner outputs $\mathcal{E}^{(m)} \in \mathbb{R}^{N_l \times N_l \times K'}$ to represent the higher-level edge representations. Specifically, $\mathcal{E}^{(m)}$ is composed of the representation of each edge, *i.e.,* $\{\boldsymbol{e}_{ij}^{(m)} \in \mathbb{R}^{K'} \mid \langle v_i, v_j\rangle \in \mathbb{E}\}$ (note that for $\langle v_i, v_j\rangle \notin \mathbb{E}$, $\boldsymbol{e}_{ij}^{(m)}$ is set as zero). Then the higher-level edge representation for edge $\langle v_i, v_j\rangle$ is computed by:

$$\boldsymbol{e}_{ij}^{(m)} = \mathrm{ReLU}\left(\mathbf{W}^{(m)}\left(\boldsymbol{v}_i^{(m-1)} \parallel \boldsymbol{v}_j^{(m-1)} \parallel \boldsymbol{e}_{ij}^{(m-1)}\right) + b^{(m)}\right), \tag{8}$$

where $\parallel$ denotes vector concatenation, $\mathbf{W}^{(m)} \in \mathbb{R}^{K' \times (2D+K)}$ is the weight matrix and $b^{(m)} \in \mathbb{R}$ is the bias, respectively. In this way, both the attributes and structure of attributed graph can be modeled for learning edge representations.

Our graph meta knowledge learner has M iterations, and the final output representations will be used as the meta knowledge for modeling ST correlations. In particular, the receptive field of each node or edge increases exponentially with respect to M. Thus, the selection of hyper-parameter M refers to how far the characteristics of nodes and edges can impact each other according to the real-world datasets. In summary, our graph meta knowledge learner can extract node and edge meta knowledge by aggregating neighbors' information, such that it can tackle the complex impacts of attributes and graph structure.

3.3 Meta Learners

As ST correlations of STGs are impacted by the characteristics of the attributed graph, we propose to learn the adjacency matrices of SC layer and kernels of TC layer from the meta knowledge of the attributed graph by meta learners. In this subsection, we introduce the meta learners for SC layer and TC layer in detail.

SC-Meta Learner. As spatial correlations are impacted by edge meta knowledge, we employ SC-meta learners to learn adjacency matrices from the edge meta knowledge $\mathcal{E}^{(M)} \in \mathbb{R}^{N_l \times N_l \times K}$, which consists of $\left\{\boldsymbol{e}_{ij}^{(M)} \in \mathbb{R}^{K} \mid \langle v_i, v_j\rangle \in \mathbb{E}\right\}$. First, it employs an FC layer to learn edge representation $\mathcal{E} \in \mathbb{R}^{N_l \times N_l \times K'}$, which is composed of $\left\{\boldsymbol{e}_{ij} \in \mathbb{R}^{K'} \mid \langle v_i, v_j\rangle \in \mathbb{E}\right\}$. Each vector $\boldsymbol{e}_{ij}$ is computed by:

$$\boldsymbol{e}_{ij} = \mathbf{W}_{\mathrm{mat}}\boldsymbol{e}_{ij}^{(M)} + b_{\mathrm{mat}}, \tag{9}$$

where $\mathbf{W}_{\mathrm{mat}} \in \mathbb{R}^{K' \times K}$, $b_{\mathrm{mat}} \in \mathbb{R}$ are learnable weights and bias, respectively. Then Equation (3) is adopted to generate the adjacency matrices, *i.e.,* $\mathcal{A} = \mathrm{DM}(\mathcal{E})$. Finally, we can use these adjacency matrices to compute SC $(\mathcal{H}, \mathcal{A}, \mathbb{W})$ according to Equation (5).

TC-Meta Learner. As temporal correlations depend on node meta knowledge, we employ TC-meta learner to generate the convolution kernels $\mathbb{K} = \left\{\mathcal{K}_1, ..., \mathcal{K}_{N_l}\right\}$ from $\mathbf{V}^{(M)} = \left[\boldsymbol{v}_1^{(M)}, ..., \boldsymbol{v}_{N_l}^{(M)}\right] \in \mathbb{R}^{N_l \times D}$. Suppose each $\mathcal{K}_i$ has N_{ker} weights. We learn all these weights by an FC layer from vector $\boldsymbol{v}_i^{(M)}$, and then reshape the output vector to the tensor shape of $\mathcal{K}_i$, which can be expressed as:

$$\mathcal{K}_i = \mathrm{reshape}\left(\mathbf{W}_{\mathrm{ker}}\boldsymbol{v}_i^{(M)} + b_{\mathrm{ker}}\right), \tag{10}$$

where $\mathbf{W}_{\mathrm{ker}} \in \mathbb{R}^{N_{\mathrm{ker}} \times D}$, $b \in \mathbb{R}$ are learnable weight and bias, respectively. After generating kernels $\mathbb{K}$, we can adopt Equation (6) to compute TC $(\mathcal{H}, \mathbb{K})$.

3.4 Searching Algorithm

In our AutoSTG, all computations are differentiable. Thus we can employ a bi-level gradient-based optimization algorithm like DARTS [15], that updates network weight parameter set Θ (including the parameters in operations, graph meta knowledge learners, and meta learners) and architecture parameter set Λ (including the scores of candidate operations) alternately. As shown in Algorithm 1, we first construct datasets, and then initialize all parameters (Line 1-2). After that, we alternately adopt a training dataset to update weight parameters (Line 4-6) and a validation dataset to update architecture parameters (Line 7-8), until the stopping criteria is met. At last, we can acquire the optimal architecture by selecting the candidate operations with the highest operation scores, and adopt the training dataset to further train the network weight parameters of this neural architecture like normal neural networks (Line 10).

Algorithm 1: Optimization algorithm of AutoSTG

input : STG data $\left[\mathbf{X}_1, ..., \mathbf{X}_{N_t}\right]$, attributes $\mathbf{V}^{(0)}$ and $\mathcal{E}^{(0)}$
1 Build $\mathbb{D}_{\mathrm{train}}$, $\mathbb{D}_{\mathrm{valid}}$ from $\left[\mathbf{X}_1, ..., \mathbf{X}_{N_t}\right]$, $\mathbf{V}^{(0)}$, and $\mathcal{E}^{(0)}$
2 Initialize Θ and Λ
3 **do**
4 Sample $\mathbb{D}_{\mathrm{batch}}$ from $\mathbb{D}_{\mathrm{train}}$
5 // For efficiency, we adopt first-order approximation
6 $\theta \leftarrow \theta - \lambda_\Theta \nabla_\theta \mathrm{L}(\Theta, \Lambda, \mathbb{D}_{\mathrm{batch}}), \forall \theta \in \Theta$ // λ_Θ is learning rate
7 Sample $\mathbb{D}_{\mathrm{batch}}$ from $\mathbb{D}_{\mathrm{valid}}$
8 $\alpha \leftarrow \alpha - \lambda_\Lambda \nabla_\alpha \mathrm{L}(\Theta, \Lambda, \mathbb{D}_{\mathrm{batch}}), \forall \alpha \in \Lambda$ // λ_Λ is learning rate
9 **until** *stopping criteria is met*
10 Get the learned architecture, and further train it using $\mathbb{D}_{\mathrm{train}}$

4 EVALUATION

In this section, we present the experimental results of AutoSTG. To support the reproducibility, we release our datasets, codes, and pretrained models on https://github.com/panzheyi/AutoSTG.

4.1 Experimental Settings

4.1.1 Task Descriptions. We conduct experiments on two real-world traffic prediction tasks, by using datasets released by [12]:

- **PEMS-BAY.** This dataset contains the traffic speed readings of the 325 sensors collected by the California Transportation Agencies Performance Measurement System.
- **METR-LA.** This dataset contains the traffic speed readings of the 207 sensors on the highway of Los Angeles County.

Table 2: Statistics of two datasets.

Dataset	PEMS-BAY	METR-LA
Time	1/1/2017-6/30/2017	3/1/2012-6/30/2012
Time interval	5 minutes	5 minutes
# timestamps	52116	34272
# nodes	325	207
# edges	5200	3312

Table 2 shows the data statistics. We adopt the GPS coordinates of sensors as node attributes and the road distance between sensors as edge attributes to build the attributed graph $\mathbb{G}$. The edge attributes are collected in both directions like [12]. We predict the next 1-hour traffic speed given the previous 1-hour traffic speed and $\mathbb{G}$. Both datasets are partitioned along the time axis into non-overlapping subsets by a ratio of 7:1:2 for training, validating, and testing.

4.1.2 Metrics. Mean absolute error (MAE) and rooted mean square error (RMSE) are adopted to evaluate our framework:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|, \qquad \text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2},$$

where N is the number of instances, $\hat{y}_i$ is the predicted value, and y_i is the ground truth.

4.1.3 Baselines. We first compare AutoSTG with the following STG prediction models for urban traffic:

- **HA**. Historical Average. We model urban traffic as a seasonal process with a one-day period and take the average of the previous seasons as the prediction results.
- **GBRT**. Gradient Boosting Regression Tree is a non-parametric statistical learning method for regression problems. For each future timestamp, we train a GBRT for its prediction, where the previous traffic speed and node attributes are given as the inputs.
- **GAT-Seq2Seq** [16]. It employs a sequence-to-sequence architecture and graph attention networks for STG prediction.
- **DCRNN** [12]. It combines diffusion convolution operations and sequence-to-sequence architecture to predict STGs.
- **Graph WaveNet** [21]. It uses graph convolution network and WaveNet to model spatial and temporal correlations, respectively. A self-adaptive adjacency matrix is also learned to discover unseen graph structures from data without any prior knowledge.
- **ST-MetaNet$^+$** [17]. This model consists of two types of meta learned structures, *i.e.,* meta graph attention network$^+$ and meta recurrent neural network$^+$, capable of capturing diverse spatial and temporal correlations of STGs, respectively.

In addition, we compare AutoSTG with two basic NAS methods:

- **RANDOM**. We randomly sample neural architectures from our search space, and train it from scratch for STG prediction.
- **DARTS** [15]. It employs continuous relaxation on candidate operations, enabling gradient-based optimization on network weight parameters and architecture parameters. In our experiments, basic DARTS framework (*i.e.,* without graph meta knowledge learner and meta learners) is employed on our search space for STG prediction.

4.1.4 Framework Settings and Training Details. In our implementation, we add ReLU activation and Batch Normalization (BN) into SC and TC, for non-linear projection and easy training. The order is ReLU-X-BN, where X denotes SC or TC layer. In addition, we employ the operation sampling technique introduced in [2] to save the GPU memory consumption in the searching process. The hyper-parameter settings of our framework contain the following parts, and we conduct grid search to find the best setting:

- We search the number of cells over $\{1, 2, 3, 4, 5, 6\}$.
- We search the number of vertices in each cell over $\{2, 3, 4, 5\}$.
- We search the number of cell hidden units over $\{16, 32, 64\}$.
- We search the number of iterations in graph meta knowledge learner over $\{0, 1, 2, 3\}$.

Our framework is tested on Ubuntu 16.04 with a single GTX 1080Ti GPU. The batch size is set as 24 for PEMS-BAY dataset and 48 for METR-LA dataset, respectively. We adopt Adam optimizer to search the architectures for 60 epochs with 64 iterations per epoch (about 12 hours in total). After that, we reinitialize the optimizer and further train the architecture for another 60 epochs (about 5 hours). In both searching and training processes, the initial learning rate is 0.01, and it reduces by a factor of 10-fold every 10 epochs.

4.2 Performance Comparison

Table 3 shows the performance of the baselines and the architectures provided by our framework. Specifically, we present the prediction performance at the next 3/6/12 timestamps as well as the overall results, which stands for the average loss over all the output timestamps. We train and test each model five times with different random seeds, and present the results in the format: "mean±standard deviation". In particular, our AutoSTG provides five architectures in each task by using different random seeds.

First, the non-deep-learning models, *i.e.,* HA and GBRT, achieve the worst performance on these two tasks, because these models only consider the human-crafted statistical features of input data. As STGs are very complex, the ST correlations related to attributed graphs cannot be fully described by human's prior knowledge, *i.e.,* human-crafted features. As a result, the non-deep-learning models have limitations on model expressiveness.

Second, GAT-Seq2Seq employs graph attention networks and sequence-to-sequence architecture to capture ST correlations. As it learns high-level ST features from data, it achieves large improvement over non-deep-learning models. Moreover, DCRNN and Graph WaveNet use diffusion convolution for capturing spatial correlations. Particularly, edge attributes (road distance) are adopted to generate diffusion matrices by a pre-defined function. Such additional knowledge can further improve the prediction accuracy. However, designing such pre-defined function depends on the prior knowledge of datasets. As prior knowledge differs in different tasks and sometimes could be very complex, it is hard to find a perfect way to use these attributes. To overcome this issue, ST-MetaNet$^+$

Table 3: Predictive performance on PEMS-BAY and METR-LA datasets.

	MAE (↓)				RMSE (↓)			
PEMS-BAY	Overall	15 min	30 min	60 min	Overall	15 min	30 min	60 min
HA	3.84±0.00	3.84±0.00	3.84±0.00	3.84±0.00	7.16±0.00	7.16±0.00	7.16±0.00	7.16±0.00
GBRT	1.96±0.02	1.49±0.01	1.99±0.02	2.61±0.04	4.48±0.00	3.21±0.00	4.51±0.01	5.76±0.02
GAT-Seq2Seq	1.74±0.00	1.38±0.01	1.79±0.00	2.26±0.01	4.08±0.01	2.94±0.01	4.10±0.01	5.22±0.04
DCRNN	1.59±0.00	1.31±0.00	1.65±0.01	1.97±0.00	3.70±0.02	2.76±0.01	3.78±0.02	4.60±0.02
Graph WaveNet	1.59±0.00	1.31±0.01	1.65±0.01	1.98±0.03	3.66±0.04	**2.75±0.01**	3.73±0.04	4.56±0.06
ST-MetaNet$^+$	1.60±0.01	1.31±0.00	1.66±0.06	1.99±0.01	3.72±0.02	2.78±0.01	3.81±0.01	4.62±0.04
AutoSTG	**1.56±0.01**	**1.31±0.00**	**1.63±0.01**	**1.92±0.01**	**3.57±0.02**	2.76±0.01	**3.67±0.02**	**4.38±0.03**
METR-LA	Overall	15 min	30 min	60 min	Overall	15 min	30 min	60 min
HA	4.79±0.00	4.79±0.00	4.79±0.00	4.79±0.00	8.72±0.00	8.72±0.00	8.72±0.00	8.72±0.00
GBRT	3.86±0.01	3.16±0.00	3.85±0.00	4.86±0.01	7.49±0.01	6.05±0.00	7.50±0.00	9.10±0.02
GAT-Seq2Seq	3.28±0.00	2.83±0.01	3.31±0.00	3.93±0.01	6.66±0.01	5.47±0.01	6.68±0.00	8.03±0.02
DCRNN	3.04±0.01	2.67±0.00	3.08±0.01	3.56±0.01	6.27±0.03	5.18±0.01	6.20±0.03	7.53±0.04
Graph WaveNet	3.05±0.01	2.70±0.01	3.08±0.01	3.55±0.12	6.16±0.03	5.16±0.01	6.20±0.03	7.35±0.05
ST-MetaNet$^+$	**3.00±0.01**	**2.65±0.01**	**3.04±0.01**	3.48±0.02	6.16±0.02	**5.11±0.01**	**6.16±0.02**	7.37±0.04
AutoSTG	3.02±0.00	2.70±0.01	3.06±0.00	**3.47±0.01**	**6.10±0.01**	5.16±0.01	6.17±0.01	**7.27±0.01**

introduces meta learning to capture the relationships between attributes and ST correlations, and achieves comparable accuracy by using much less trainable parameters.

Different from these baselines, our AutoSTG is capable of automatically learning neural architectures to capture ST correlations, and achieves stable and competitive prediction accuracy compared with these expert-designed networks. Especially in PEMS-BAY dataset, it improves the prediction accuracy over 2% than the previous state-of-the-art results in terms of both MAE and RMSE.

Notice that the architectures provided by our AutoSTG have 528k±36k and 496k±31k trainable parameters (mean±std) for PEMS-BAY and METR-LA dataset, respectively, more than that of DCRNN (372k), Graph WaveNet (297k), and ST-MetaNet$^+$ (162k). However, we cannot improve the accuracy of these baselines by adding more parameters. This fact can also somehow illustrate the effectiveness of our searching algorithm. And in our future work, we will try to control the trainable parameters in NAS framework.

4.3 Ablation Studies

In this subsection, we conduct ablation studies for AutoSTG. First, we evaluate the effectiveness of our candidate operation set. We compared our framework with three variants: 1) **w/o SC & TC**, which replaces SC and TC with FC operation in our candidate operation set; 2) **w/o SC**, which removes SC from the candidate operation set; and 3) **w/o TC**, which removes TC from the candidate operation set. As shown in Figure 6, both SC and TC can improve the prediction accuracy in modeling STGs. In particular, SC makes more contributions to the final results, whereas TC shows less improvement. The reason is that the temporal pooling layers can somehow capture temporal correlations by aggregating information in the temporal domain. In contrast, spatial correlations can only be captured by our SC operation. As a result, removing SC operation makes the performance degrade significantly.

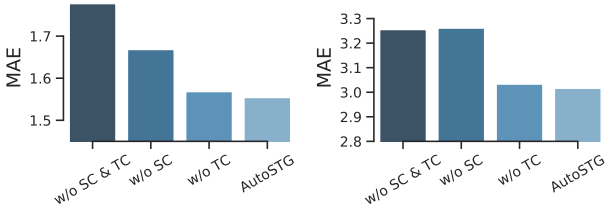


Figure 6: Ablation studies on candidate operations.

Second, we conduct experiments to verify the effectiveness of each framework component by using the following NAS methods: 1) **RANDOM**, which samples an architecture from our search space and then trains this architecture for prediction; 2) **DARTS**, which removes meta learning method in our framework; and 3) **w/o graph**, which replaces the graph representation learning network (*i.e.,* the graph meta knowledge learner) with the basic fully connected networks for representation learning of nodes and edges like [16]. As shown in Figure 7, our AutoSTG performs significantly

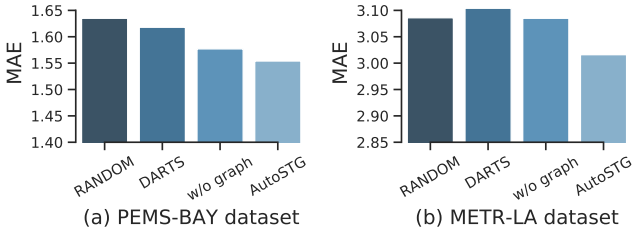


Figure 7: Ablation studies on framework components.

better than these variants that removes architecture search process (RANDOM), meta learning technique (DARTS), or graph representation learning network (w/o graph), illustrating that all these components are effective to model STGs.

4.4 Evaluation on Framework Settings

AutoSTG has several hyperparameters, including the number of cells, the number of vertices in each cell, the number of hidden units of each cell, and the number of iterations in graph meta knowledge learner. To investigate the robustness of our AutoSTG, for each hyperparameter, we present the prediction accuracy under different choices of it by fixing the other hyperparameters.

Figure 8 presents the evaluation results of these hyperparameters. As shown in Figure 8(a), Figure 8(b), Figure 8(e), and Figure 8(f), the number of cells and the number of vertices in each cell are related to the number of network operations in candidate neural architectures. Increasing these values can promote network expressiveness at first, but then lead to overfitting as too many trainable parameters are introduced into candidate architectures. Likewise, as shown in Figure 8(c) and Figure 8(g), increasing the number of hidden units (*i.e.,* feature dimension) can also promote the capability of network structures, but too large dimension would degrade the performance.

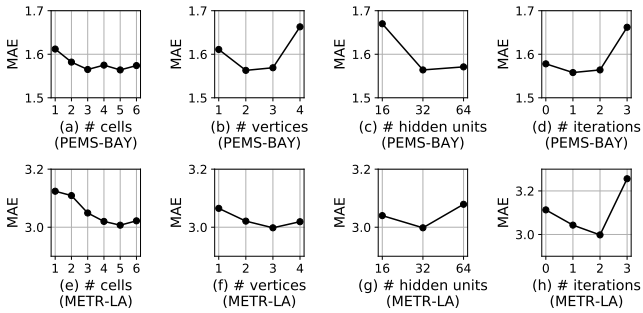


Figure 8: Studies on hyperparameters.

Different from the above hyperparameters, the number of iterations in graph meta knowledge learner does not significantly impact the number of trainable parameters. For example, in our experiments, there are only several thousands of parameters in graph meta knowledge leaner, much less than the parameters in network architecture. However, as shown in Figure 8(d) and Figure 8(h), adopting more iterations can also improve the prediction results. The reason is that this hyperparameter reflects the receptive field of node and edge characteristics. Aggregating neighbors' information for nodes and edges can enlarge such receptive field and tackle graph structure in advance, helping AutoSTG learn better representations of attributed graph. Nevertheless, as simply stacking amounts of graph convolutional network is hard to be optimized, too many iterations would also lower the accuracy. Therefore, that is a trade-off to set this hyperparameter in real-world tasks.

4.5 Empirical Studies of Learned Architectures

In general, the properties of datasets are different, and accordingly the neural architectures diverse. To verify it, first we study the properties of PEMS-BAY and METR-LA dataset. As the distributions show in Figure 9(a), vehicles often keep high speed in PEMS-BAY dataset, while the speed in METR-LA dataset is much slower, indicating more traffic jams. Intuitively, when traffic congestion occurs, the traffic of a node has a larger impact on the nearby nodes' traffic, whereas if vehicles keep high speed, the traffic speed of nodes are less correlated. As shown in Figure 9(b), we present the distributions of Pearson correlations between all pairs of nodes, and it illustrates

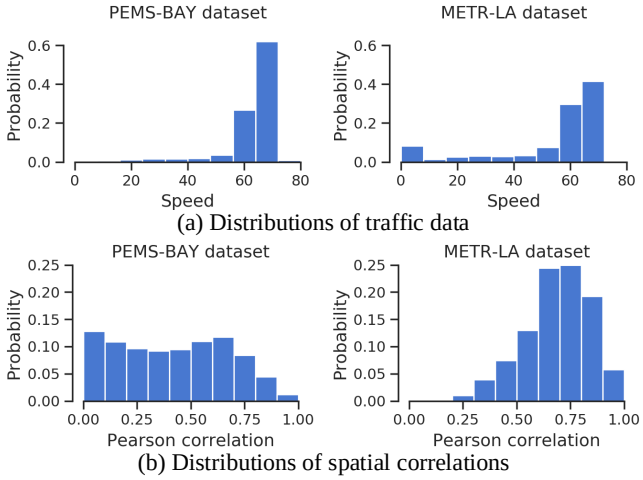


Figure 9: Statistics of PEMS-BAY and METR-LA dataset.

that the traffic data of PEMS-BAY dataset is much less correlated than that of METR-LA dataset. Thus, empirically we need more SC operations for predicting METR-LA dataset to achieve a larger receptive field and capture long-distance spatial correlations.

Next, as shown in Figure 10, for each dataset, we learn 5 architectures using different random seeds and plot the number of SC operations of these architectures. This figure shows that the architectures for predicting METR-LA dataset contain nearly double amounts of SC operations than that of PEMS-BAY dataset, which verifies our assumption. Therefore, it demonstrates that our AutoSTG can find customized architectures according to dataset properties.

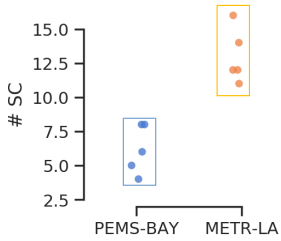


Figure 10: The number of SC in the learned architectures.

5 RELATED WORK

Deep Learning for Spatio-Temporal Graph. Deep learning was widely adopted in STG prediction. For example, [4, 8, 20, 21, 23] combined convolutional neural networks and graph convolutional neural networks for modeling STGs, while some other studies [3, 12, 28] employed recurrent neural networks and graph convolutional neural networks for STG prediction. Moreover, [6, 7, 13, 19, 29] studied the usage of attention mechanisms to capture ST correlations. Recently, [16, 17] leveraged meta learning method to model the inherent relationships between graph attributes and diverse ST correlations, and [27] proposed to learn graph structures from dynamic states. These works provide insights (*e.g.,* possible network structures) for STG prediction. However, designing models for specific tasks requires substantial domain knowledge and large amounts of expert efforts. By contrast, our work can automatically search

promising neural architectures for STGs, which is more efficient and cost-saving.

Neural Architecture Search. Many works have explored neural architectures for grid-based data (*e.g.,* image) or sequential data (*e.g.,* natural language). At first, reinforcement learning was adopted to search neural architectures [14, 18, 32]. Later, some one-shot algorithms tried to generate the weights of the sampled architectures by hypernetworks [1, 24]. By contrast, [15] proposed continuous relaxation on candidate operators, enabling gradient-based optimization on architecture parameters. Besides these conventional data types, [11] proposed to search architectures for grid-based ST data and [5, 31] employed NAS to model graph structure data. However, these works cannot be directly applied to predict STGs, as it has a different search space to model ST correlations, and such correlations are affected by the meta knowledge of attributed graph. To the best of our knowledge, we are the first to study NAS algorithm for STG prediction.

6 CONCLUSION

In this paper, we propose a novel NAS framework, entitled AutoSTG, for automated STG prediction. It employs SC and TC operations in search space to model spatial and temporal correlations, respectively. To capture the relationships between attributed graph and ST correlations, we use a graph meta knowledge learner to extract the graph characteristics, and then apply meta learning to generate the diffusion matrices of SC layers and the kernels of TC layers from the learned characteristics. We conduct extensive experiments on two real-world benchmark datasets to demonstrate that our AutoSTG can find promising architectures and achieve significant prediction accuracy. In the future, we will extend our framework to a broader set of STG prediction tasks, and further improve the complexity and efficiency of the searched architectures.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (2019YFB2101801), the National Natural Science Foundation of China (No. 62076191, 72061127001 and 61772333), and the Beijing Nova Program (Z201100006820053).

REFERENCES

[1] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. 2017. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344* (2017).
[2] Han Cai, Ligeng Zhu, and Song Han. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332* (2018).
[3] Weiqi Chen, Ling Chen, Yu Xie, Wei Cao, Yusong Gao, and Xiaojie Feng. 2019. Multi-range attentive bicomponent graph convolutional network for traffic forecasting. *arXiv preprint arXiv:1911.12093* (2019).
[4] Rui Dai, Shenkun Xu, Qian Gu, Chenguang Ji, and Kaikui Liu. 2020. Hybrid Spatio-Temporal Graph Convolutional Network: Improving Traffic Prediction with Navigation Data. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3074–3082.
[5] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2019. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. *arXiv preprint arXiv:1904.09981* (2019).
[6] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 922–929.
[7] Suining He and Kang G Shin. 2020. Towards fine-grained flow forecasting: A graph attention approach for bike sharing systems. In *Proceedings of The Web Conference 2020*. 88–98.
[8] Rongzhou Huang, Chuyin Huang, Yubao Liu, Genan Dai, and Weiyang Kong. [n.d.]. LSGCN: Long Short-Term Traffic Prediction with Graph Convolutional Networks. ([n. d.]).
[9] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).
[10] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[11] Ting Li, Junbo Zhang, Kainan Bao, Yuxuan Liang, Yexin Li, and Yu Zheng. 2020. Autost: Efficient neural architecture search for spatio-temporal prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.
[12] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2017. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926* (2017).
[13] Yuxuan Liang, Songyu Ke, Junbo Zhang, Xiuwen Yi, and Yu Zheng. 2018. Geoman: Multi-level attention networks for geo-sensory time series prediction. In *IJCAI*. 3428–3434.
[14] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 19–34.
[15] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
[16] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. 2019. Urban traffic prediction from spatio-temporal data using deep meta learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1720–1730.
[17] Zheyi Pan, Wentao Zhang, Yuxuan Liang, Weinan Zhang, Yong Yu, Junbo Zhang, and Yu Zheng. 2020. Spatio-Temporal Meta Learning for Urban Traffic Prediction. *IEEE Transactions on Knowledge and Data Engineering* (2020).
[18] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* (2018).
[19] Xian Wu, Chao Huang, Chuxu Zhang, and Nitesh V Chawla. 2020. Hierarchically structured transformer networks for fine-grained spatial event forecasting. In *Proceedings of The Web Conference 2020*. 2320–2330.
[20] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. 2020. Connecting the Dots: Multivariate Time Series Forecasting with Graph Neural Networks. *arXiv preprint arXiv:2005.11650* (2020).
[21] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph wavenet for deep spatial-temporal graph modeling. *arXiv preprint arXiv:1906.00121* (2019).
[22] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2018. SNAS: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926* (2018).
[23] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875* (2017).
[24] Chris Zhang, Mengye Ren, and Raquel Urtasun. 2018. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749* (2018).
[25] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.
[26] Junbo Zhang, Yu Zheng, Dekang Qi, Ruiyuan Li, Xiuwen Yi, and Tianrui Li. 2018. Predicting citywide crowd flows using deep spatio-temporal residual networks. *Artificial Intelligence* 259 (2018), 147–166.
[27] Qi Zhang, Jianlong Chang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2020. Spatio-Temporal Graph Structure Learning for Traffic Forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 1177–1185.
[28] Weijia Zhang, Hao Liu, Yanchi Liu, Jingbo Zhou, and Hui Xiong. 2020. Semi-Supervised Hierarchical Recurrent Graph Neural Network for City-Wide Parking Availability Prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 1186–1193.
[29] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. 2020. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 1234–1241.
[30] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. 2014. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 3 (2014), 1–55.
[31] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. 2019. Auto-GNN: Neural Architecture Search of Graph Neural Networks. *arXiv preprint arXiv:1909.03184* (2019).
[32] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).