Efficient Spatio-Temporal Randomly Wired Neural Networks for Traffic Forecasting

Li Song[1] †, Kainan Bao[2] †, Songyu Ke[3] ‡, Chunyang Li[4] ‡, Junbo Zhang[1]* ‡, Yu Zheng[1] ‡
[1]JD iCity, JD Technology, Beijing, China
[2]Southwest Jiaotong University, Chengdu, China
[3]Shanghai Jiao Tong University, Shanghai, China
[4]Xidian University, Xi'an, China
†{song200626,baokainan123}@gmail.com
‡{songyu-ke,lichunyang_1,msjunbozhang,msyuzheng}@outlook.com

***Abstract*—Designing neural architecture is very important for traffic forecasting. In the past decade, it mainly depends on human experts. Recently, neural architecture search (NAS) has been leveraged to automatically find the optimal candidates for diversity spatio-temporal forecasting applications. However, it still depends on experts' experiments to narrow down the search space to make a trade-off between accurate forecasting and the high computational cost. In this paper, we try to generate the neural architecture for spatio-temporal forecasting. We propose a novel and efficient spatio-temporal randomly wired neural network generation method, termed ST-RWNet. ST-RWNet is not a fixed neural architecture. It can produce a bunch of spatio-temporal neural networks. ST-RWNet includes a spatio randomly neural network generator, a temporal randomly neural network generator, and a randomly bipartite neural network generator. Those neural network generators are based on random graphs, which control the attributes of neural networks, *e.g., the number of pathways, the length distribution of pathways.* The spatio and temporal neural network generators are designed to capture the spatio and temporal dependencies respectively. And the randomly bipartite neural network generator is designed to fuse the complicated intermediate spatio-temporal features. Extensive experiments on real-world traffic forecasting datasets have demonstrated that the performance is competitive with state-of-the-art methods. And our method achieves up to $6\times$ speed up compared to other NAS-based methods.**

***Index Terms*—Spatio-temporal forecasting, urban computing, neural architecture search**

I. Introduction

Designing neural network architecture is of great importance for traffic forecasting due to complicated spatio-temporal (ST) dependencies. As shown in Fig. 1, to predict the traffic flow of location S_1, both the spatio and temporal dependencies should be taken into consideration. This means that the traffic flow has a complicated relationship with nearby timestamps. And the traffic flow at one point S_2 is influenced by others, *e.g.*, S_1 and S_3. Many deep learning methods have been designed to capture spatio-temporal dependencies so that they can make accurate forecasting.

In the past decade, spatio-temporal forecasting neural networks are mainly designed by human experts [1], [2]. However, it is non-trivial to directly deploy it to urban platforms since it depends on expert experiments to redesign a city-aware

* Junbo Zhang is the corresponding author.

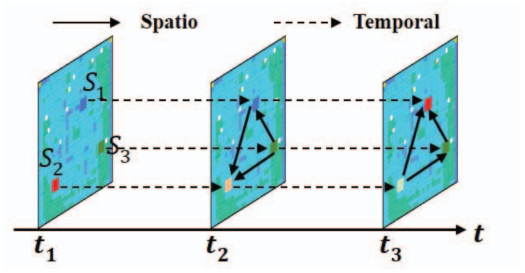


Fig. 1. Spatio-temporal correlations.

model for different cities to capture the spatial and temporal correlations. To incorporate the tedious and monotonous work, spatio-temporal neural architecture search (ST-NAS) methods [3]–[5] have been used to automatically find the optimal architecture. However, those ST-NAS methods still have the following limitations.

First, the computation cost isn't affordable. ST-NAS methods find the best architecture by solving a combinatorial optimization problem through a very large search space. Since evaluating the performance of a new neural network is time-consuming, a lot of work focuses on how to speed up the training processing of NAS. Some approximate methods *e.g., weight sharing* can speed up the searching process but also harm the performance of the final neural architecture, which may lead to an inferior solution. As [6] has presented, the rank of the architecture's performance searched by weight-sharing NAS is not consistent with the rank of the architectures trained from scratch in a limited search space.

Second, the search space is constrained. The search space of current ST-NAS methods heavily depends on domain knowledge and expert efforts. Spatio operations and temporal operations designed by human experts are the main components of existing search space, which may omit the potential operations with high effectiveness.

Third, ST features from intermediate levels are not comprehensively fused. Many factors influence the traffic flow [1], [2]. On the one hand, we should consider the spatio-temporal dependencies of historical traffic flow. The temporal correlations

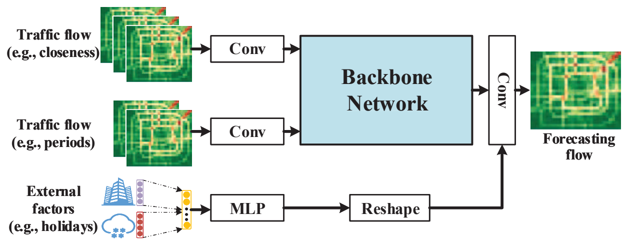


Fig. 2. The framework of spatio-temporal traffic forecasting.

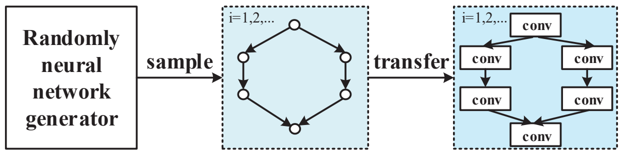


Fig. 3. Random neural network generator.

include closeness, periods, and trends. The spatial correlations consist of long-term and short-term dependencies between different regions. On the other hand, we should carefully fuse the intermediate-level features from distinct components.

To corporate the above three challenges simultaneously, we propose a novel and efficient spatio-temporal randomly wired neural network (**ST-RWNet**) for traffic forecasting. The whole framework of spatio-temporal forecast is shown in Fig. 2, we take the closeness and periods of traffic flow to forecast the next timestamps. The detail of closeness and periods of traffic flow can be found in [2]. We propose randomly neural network generator to design the architecture of the backbone network. As illustrated in Fig. 3, we sample a graph from the randomly neural network generator. Then, we transfer the topology of the sampled graph to a neural architecture using the method proposed in [7]. We design spatio randomly neural network generator and temporal randomly neural network generator to capture the spatio and temporal dependencies, respectively. Besides, we introduce the bipartite graph to fuse information from different views. The connection of the bipartite graph is leveraged to fusion intermediate features from different components automatically. Since those randomly wired neural networks are generated by random graphs which imposed prior knowledge from human beings, *e.g., the number of pathways, the length distribution of pathways*, it is likely to capture certain spatio-temporal patterns. The structure of random bipartite graphs allows the intermediate features to transfer from one component to the others. Compared with the hand-crafted neural architecture, the proposed method is flexible in determining the neural architecture. Extensive experiments on 4 real-world traffic forecasting tasks have demonstrated the efficiency of our proposed method. Concretely, we achieve up to 6× speed up compared with the state-of-the-art NAS-based methods and the performance is competitive.

We summarize our contributions to this article as follows:

- We propose a novel model of spatio-temporal randomly wired neural networks. The random graph generators encoding the prior knowledge from human beings are leveraged to generate the neural network architectures. It has the potential to alleviate the dependencies of human experts and reduce the computation cost.
- We design the spatial and temporal random neural network generator for spatial and temporal correlation. Those random graph generators are more efficient to design the neural architecture. Besides, the bipartite network generator is proposed to automatically fuse the intermediate features. The way is more flexible than traditional human-designed fusion methods, e.g., early fusion, later fusion, and central fusion, which allows the intermediate level features to exchange information.
- We conduct experiments on real-world traffic forecasting tasks to demonstrate the effectiveness and efficiency of the proposed method. The search speed is up to 6× speedup compared to current NAS methods, which shows a substantial superiority over existing traffic forecasting applications. And the performance of our proposed method is competitive with those methods and more stable on different datasets.

II. Preliminaries

In this section, we first introduce the definitions of the traffic forecasting problem and the neural architecture search. Then, we give an overview of the random graph.

A. Definitions and Problem Statements

Definition 2.1 (Traffic flow): We partition a city into an $I \times J$ grid map with equal step size on the longitude and latitude, where a grid denotes a region. We also partition the timeline with a fixed time interval l. Then, we count the number of inflow and outflow of each region at the t^{th} time interval by counting the number of entities that enter in or pass through this region. The traffic flow $X_t \in \mathbb{R}^{2\times I\times J}$ at the t^{th} time interval is formatted according to the spatial location of each region. Finally, we stack the traffic flow at each time interval to get the traffic flow of the city $\mathcal{X} \in \mathbb{R}^{L\times 2\times I\times J}$, where L is the total number of time intervals.

Definition 2.2 (Traffic flow forecasting): The goal of traffic forecasting is to predict the future traffic flow given previously observed traffic flow.

$$[X_1, X_2, ..., X_T] \mapsto X_{T+1}, \tag{1}$$

where T is the timestamps of the historical traffic flow.

Definition 2.3 (Neural architecture search, NAS): NAS for spatio-temporal forecasting aims to find the best architecture α^* for a given traffic flow forecasting dataset by minimal the loss on the validation dataset $\mathbb{D}_{val}$. By the way, network parameters w of the architecture are optimized by the training dataset $\mathcal{D}_{train}$:

$$\begin{aligned} \alpha^* &= min_\alpha \mathbb{L}(w^*(\alpha), \alpha, \mathbb{D}_{val}) \\ s.t. w^*(\alpha) &= \arg\min_w \mathbb{L}(w, \alpha, \mathbb{D}_{train}), \end{aligned} \tag{2}$$

where $\mathbb{L}$ is the loss function.

B. Graph and Random Graph

A graph G consists of a node-set V and an edge-set E, represented as $G(V, E)$. The graph is widely used to capture the relationship between entities, such as road networks. Random graphs are widely researched in graph theory. The definition of random graph encodes the prior knowledge (*e.g.*, small world) from human beings. The followings are three widely used random graphs.

a) Erdős-Rényi(ER): The ER model [8] defines the probability (P) of the connection between two nodes. There is a simple way to generate the ER graph with parameter P. Given a complete graph, for each edge, we keep this edge with probability P, otherwise, we drop this edge.

b) Barabási-Albert(BA): The BA model [9] has a parameter M, which controls the attribute of edges. Given an initial node-set with $M(1 <= M < N)$ nodes, the BA model sequentially adds a new node and builds M edges between the node and the previous node-set with a probability according to the degree of previous nodes, where N is the total number of nodes.

c) Watts-Strogatz(WS): The graph generated by WS model is a small-world graph [10]. To generate a graph, we start with a ring lattice and rewire some of the edges with a probability P.

We use those random graphs as inputs for our spatio randomly neural network generator and temporal randomly neural network generator.

III. Methodologies

In this section, we first introduce the framework for spatio-temporal forecasting. Then, we describe the detail of our proposed method spatio-temporal randomly wired neural networks (**ST-RWNet**).

A. Framework

We follow the basic framework used in [1], [2]. As shown in Fig. 2, the inputs of the network are the historical traffic flow and external factors. There may be many input traffic flows extracted along the time axis with temporal semantic meanings, such as, closeness, which takes the nearby traffic flow less then t_c steps with time stride l_c equals 1, that is $[X_{t-t_c*l_c+1}, ..., X_t]$, periods, which takes the traffic flow with a fixed periods l_p with t_p timestamps, that is $[X_{t-t_p*l_p}, ..., X_{t-t_p+1}]$.

In this paper, our main work is to design the backbone network using random graph theory to capture spatio-temporal dependencies, as shown in Fig. 3. We first propose the spatio and temporal randomly wired neural network generator to capture the spatial and temporal dependencies, respectively. Furthermore, to leverage features extracted from intermediate levels, we design the random bipartite graph generator to automatically fuse intermediate level features for spatio-temporal forecasting problems. Finally, we transfer the whole graph to spatio-temporal randomly wired neural networks (ST-RWNet). The network is trained from scratch for a given dataset by minimizing the mean squared error loss. In the following section, we will give the detail of each component.

B. Spatio Randomly Neural Network Generator

The motivation of the graph generator is that the design of neural networks for ST forecasting resorts to a paradigm, which includes two steps: 1) determine the connection of each component or node of the backbone network, such as ResNet [11] with $x + \mathcal{F}(x)$ where x and $\mathcal{F}$ represent two nodes and there is a connection between the two nodes. 2) determine the operation type of each node of the backbone network, such as convolution with kernel size equal to 3 or even more complex operator as ResPlus [2]. In the recent NAS framework, the search space can also be considered as an instance of this paradigm in which the operation of each step or layer takes two previous layers' outputs as inputs and produce the high-level features as outputs. So we can use a graph to generate a neural network by offering each node with an operation and taking each edge as a data flow. The edges transport data from nodes with low-level features to other nodes. The nodes aggregate data from others and generate features. Instead of designing a neural network manually or searching for the best architecture from a huge space with heavy computation cost, we use the graph theory to generate a topology which encodes human prior knowledge and then transfer it into a neural network.

We first introduce the rules of graph generator that transfer a graph to a neural network. Let $G(V, E)$ be a graph with $|V|$ nodes and $|E|$ edges. A random graph can be generated using the random graph algorithms as described in Section II. Concretely, we transfer the undirected graph to a directed acyclic graph (DAG) by arranging each edge in a direction from the small label to the big one, see Fig. 4. As a neural network block commonly has one input node and one output [12], we add two additional nodes, source node 0 and target node $|V| + 1$. Source node will copy the input features to all nodes whose in-degree is zero, see Fig. 4(b), from $0 \mapsto 1, 2, 3$. The target node will collect all features from nodes whose out-degree is zero, see Fig. 4(b), from $8, 9, 10 \mapsto 11$. Besides, we add a deep learning operation at each node to transfer the DAG to a neural block. Node operations are defined as Fig. 5. The node operation can be divided into three stages, that is aggregation, transformation, and distribution. Aggregation operations collect all features from the corresponding previous input nodes and use a weighted sum operator to integrate information. Transformation operations generate new high-level features with the regular convolution. And then, distribution operations seed the new features to its successors.

C. Temporal Randomly Neural Network Generator

The mechanism of the temporal random graph generator is similar to the spatial random graph generator. To capture the temporal dependencies, we set each node of the graph as a timestamp and the edge between two nodes determines the direction of information flow. Without loss of generality, we also let $G(V, E)$ be a graph with $|V|$ nodes and $|E|$

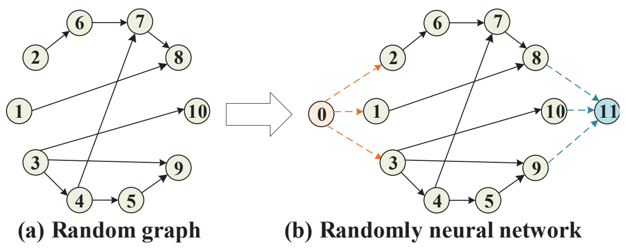


Fig. 4. Spatial random graph and randomly wired neural networks.

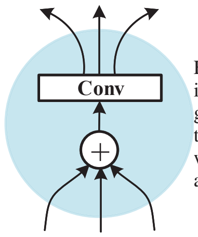


Fig. 5. Node operations. Node operations illustrate the method we transfer a random graph to a neural network. We first aggregate the information from those input nodes with a weighted sum. Then, the convoluted features are distributed to the output nodes.

edges. And the transformation from random graphs to the computation neural networks is the same as the spatial random graph generator. The difference is that the number of nodes of the temporal random graph generator equals the number of the input timestamps. As shown in Fig. 6, the input of each node is initialized with the previous spatial features generated with the spatial random graph generator. The edge determines the way how to fuse features from different timestamps. For instance, node 2 receives the spatial state from node 1 and mixes it with its spatial state using a weighted sum. After that, the fused state is sent to the successors. Finally, node 8 summarizes the spatial states from node {6,7} and its spatial states. The new states are generated by passing the state of each node to the next layer.

D. Randomly Bipartite Neural Network Generator

To fusion intermediate features, we propose to transfer a bipartite graph to a bipartite neural network. Bipartite graphs are widely used in the real world, *e.g.* the user-item network in recommendation systems [13], [14]. A bipartite graph consists of two node-sets U, V and an edge-set E, as Fig. 7(a). In this paper, node-sets indicate the intermediate features from

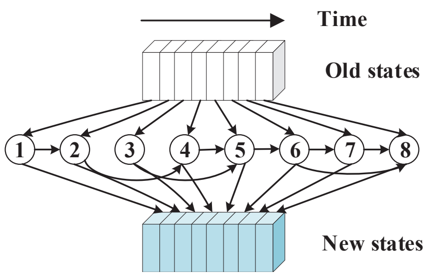


Fig. 6. Temporal randomly wired neural networks.

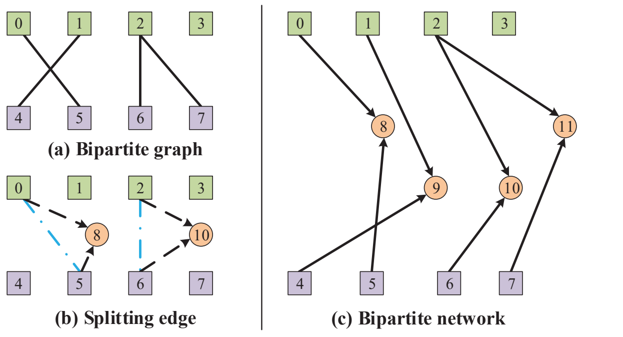


Fig. 7. Bipartite network and randomly wired fusion neural network. Green square nodes and purple square nodes indicate the intermediate level features generated from different views. The orange circle nodes are the fusion node operations.

different views (*e.g.*, closeness, periods, and trend). The edges (u, v) between the two node-sets indicate the way how to fuse those intermediate features.

To transfer a bipartite graph into a neural network block, we introduce the random bipartite graph generators. Different from a random graph, we cannot turn a bipartite graph into a neural network block directly become for the following reasons. 1) The input nodes of a bipartite graph take semantic meanings, which carry the feature information from each domain. 2) The edges of a bipartite also take semantic meanings, which decide the structure of the fusion layers that are different from the random graph situation.

To build a universal framework, we borrow the idea of splitting from graph theory. Practically, for a specific edge $(u, v), u \in U, v \in V$, we first remove the edge from the graph, then add a new node t and two edges $(u, t), (v, t)$ to the graph. For instance, the raw edge solid blue line $(0, 5)$ is removed, and two edges dotted black line $(0, 8), (5, 8)$ are added, as shown in Fig. 7(b). We split all edges and transfer the bipartite graph to a random graph, as plotted in Fig. 7(c). After that, we extend to a random bipartite graph generator by sampling a random bipartite graph.

E. Spatio-temporal Randomly Wired Neural Networks

To corporate with the traffic forecasting problem, we integrate a random graph neural network and a random bipartite graph neural network to generate the backbone network. Instead of stacking spatio-temporal blocks to generate intermediate-level features as [1], [2], we generate the architecture of each branch by random graph generators. As illustrated in Fig. 8, we first sample two random graphs from the random graph generators. And then, we transfer random graphs to a randomly wired neural network for each view. After that, we extract the feature from each node as the intermediate features. And we use a bipartite graph generator to decide the way how to fuse those intermediate features. Overall, we use the input nodes to process the input traffic flow from different views, (*e.g.*, closeness, periods, and trends). The output of the backbone network is fused with optional external factors

TABLE I
DATASETS

Dataset	Map Size	Time Span	l^1	L^2
BikeNYC	(21, 10)	4/1/2014- 9/30/2014	1h	4392
TaxiGY	(20, 24)	10/1/2018 - 5/26/2019	1h	5578
TaxiJN	(16, 32)	9/1/2017 - 1/30/2018	1h	3589
CrowdBJ	(32, 32)	9/1/2017 - 11/30/2017	1h	2184

[1] l indicates time interval.
[2] L indicates the total number of time intervals.

to make forecasting. The whole network is trained by back-propagation.

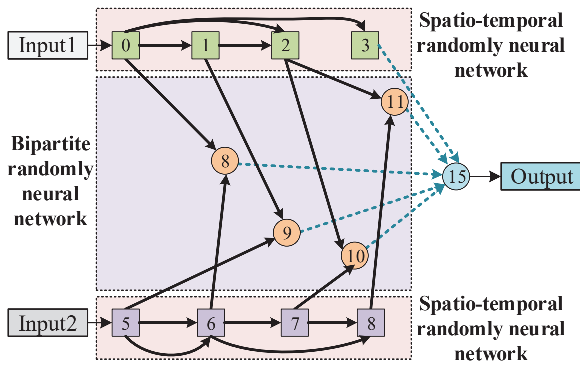

Input1
0
1
2
3
Spatio-temporal randomly neural network
11
Bipartite randomly neural network
8
15
Output
9
10
Input2
5
6
7
8
Spatio-temporal randomly neural network

Fig. 8. Spatio-temporal randomly wired neural networks.

IV. EXPERIMENTS

A. Experimental Settings

1) Task Descriptions.: We conduct experiments on 4 real-world traffic forecasting datasets. The basic information of traffic flow datasets is shown in Table I. The inputs are extracted traffic flow data and external factors following [1]. We divide the map into a $I * J$ grid size, and then we count the transmit inflow and outflow of each grid during each time interval. Suppose the history data $\mathbb{X} \in R^{L \times f \times I \times J}$, where L is the length of history timestamps and f equals 2 indicating inflow and outflow of each grid. The sample generated at time t is formatted as $X_t \in R^{l \times f \times I \times J}, Y_t \in R^{t \times f \times H \times W}$, where l, t are the length of history data and predict target. Moreover, the external factors consist of holidays, weather conditions, and POI. Our goal is to predict the inflow and outflow of each grid in the next timestamp. The inflow and outflow of each grid for 1 hour are collected. We split each dataset into the train set, validation set, and test set. The validation data and test data are the data of the 2 weeks starting from the last four weeks and the data of the last 2 weeks respectively.

2) Metrics.: We use rooted mean squared error (RMSE) and mean absolute error (MAE) to evaluate the forecasting results:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}, MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|, \tag{3}$$

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

3) Baselines.: We evaluate our method with three kinds of models, including traditional forecasting models, deep learning forecasting models designed by human experts for spatio-temporal data, and the NAS-based models adapted to spatio-temporal data.

Deep learning forecasting models include the following models:

- **HA.** HA uses the mean of traffic flow of previous timestamps to make forecasting for each grid region.
- **GBRT.** To make forecasting, we use the previous traffic flow as input to train a GBRT [15].
- **ST-ResNet.** ST-ResNet [1] is proposed to predict the traffic flow. The model has three branches closeness, periodicity, and tendency. Late fusion is utilized to fuse different information to predict traffic flow in the next timestamp.
- **DeepSTN.** [2] is different from the previous late fusion methods. Early fusion is utilized to fuse low-level data information. So only one branch is stacked to extract high-level features for prediction.
- **ConvLSTM.** ConvLSTM [16] extends the traditional recurrent neural network LSTM by replacing the matrix multiplication with convolution operation to capture the spatio-temporal correlations simultaneously.
- **ST3DNet.** ST3DNet [17] adopts 3D convolutions and residual units to extract features from both spatial and temporal dimensions.
- **CentralNet.** [18] exploits a hybrid fusion method. An additional network is used to fuse features extracted from each feature extractor. The additional network fuse features at each level, and the low-level feature is fed into high-level fusion components.

The NAS-based models for spatio-temporal forecasting consist of the following models:

- **DARTS.** We adapt DARTS [19], a gradient-based optimization strategy, to spatio-temporal forecasting. The search space includes four operations, that is, standard convolution operation with kernel size 3 and 5, identity, and zero connections.
- **EPNAS.** We adapt EPNAS [5], an efficient progressive NAS method, to spatio-temporal forecasting by searching the backbone network.
- **AutoST.** [3] is a NAS-based method. The feature is processed with early fusion and then fed into a NAS framework to search for the best architecture.
- **RandomNAS.** RandomNAS [20] uses a random search strategy to determine the best architecture by evaluating the performance of each neural network through the validation dataset.

4) Framework Settings and Training Details.: In our experiments, we implement random graphs by a python tool *networkx*[1]. We use the WS random graph generator to generate

[1]https://networkx.org/

the backbone architecture of spatio and temporal randomly neural network. For all datasets, we search the number of nodes for backbone over $\{4, 6, 8, 10, 12\}$. And we fix the probability P of the connection equals 0.75. We set t_c equals 3 and t_p equals 1 to extract the historical traffic flow. For models with only one branch, we use early fusion to integrate information. For models with two branches, the input of each branch is set to closeness and periods, respectively. We split the whole dataset into train, validation, and test datasets by fixing the size of validation and test size to 336, which includes the last two weeks and the next two-week traffic flow.

Our experiments are conducted on CentOS 7.0 with a single Tesla V100 GPU. The batch size is set to 8 for all datasets. The model is optimized by Adam [21] with an initial learing rate 0.0001. We train ST-RWNet with 200 epochs with an early stopping monitor the validation loss with patient equals to 10 epochs. We run 10 rounds and report the mean and standard deviation of RMSE and MAE on the test dataset.

B. Results

1) Performance Comparison: We first conduct experiments to validate the traffic forecasting performance. As shown in Table II, we test each model ten times with different random seeds, and report the mean and standard deviation of metrics.

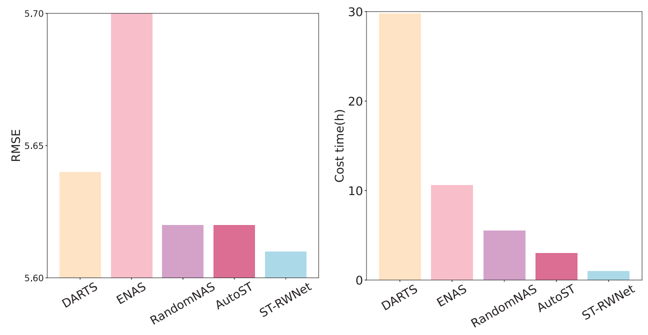
First, the non-deep-learning models, *i.e.*, HA and GBRT, achieve the worst performance on all tasks because these models only consider the human-crafted statistical features of historical traffic flow. Since spatio-temporal correlation plays an important role in traffic forecasting, those non-deep-learning models have limitations on model expressiveness.

Second, the rank of expert-designed deep learning models is not consistent over all tasks. We observe that ST-ResNet performs better than other deep learning models on TaxiGY and TaxiJN. The reason behind this may be that the grid size of those two datasets is smaller than CrowdBJ, which is 32×32. It is more difficult to capture the long-term dependencies of convolution operations.

Third, the way how to fusion information also plays a vital role in spatio-temporal traffic forecasting. It is non-trivial to determine the fusion component by early fusion, later fusion, and central fusion. Though CentralNet introduces the fusion operation at each layer, it may result in information unbalance between different views.

Fourth, it is difficult to find a good architecture for NAS methods. Since approximate strategies are applied to existing NAS methods, no NAS method performs consistently better than others. Another reason is that we cannot keep the search space of all NAS methods the same, NAS methods may fall to a local minimum. We also observe that the RandomNAS method is worse than existing NAS methods, which demonstrates the complexity of traffic forecasting.

Finally, we observe that our proposed model ST-RWNet is competitive with expert-designed models and NAS methods. The average rank of ST-RWNet is relatively high than others. That indicates that the ST-RWNet has a more promising application in the real world under the limited computation resources.



(a) Performance on BikeNYC (b) Time on BikeNYC

Fig. 9. Computation time and performance comparison.

2) Complexity Comparison: In this section, we investigate the computation time and performance comparisons between ST-RWNet and existing NAS models. We count the time cost of the training and test stage of NAS-based methods. We conduct experiments on BikeNYC dataset. We run each model ten times and report the average computation cost in hours and the RMSE performance. The results are shown in Fig. 9.

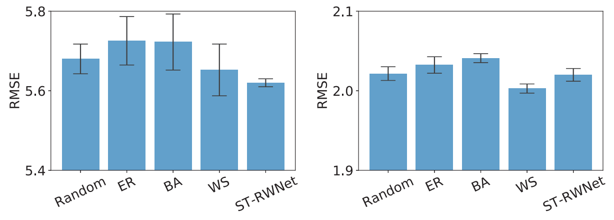
One can observe that our method is more efficient than existing NAS-based search methods. That reduces the computation cost with a substantial margin compared with the DARTS, EPNAS, and AutoST methods. We achieve about $6\times$ speed up than the ordinary DARTS strategy. The computation cost is about $2\times$ fast than AutoST, whose search space is delicately designed for spatio-temporal forecasting with human experiments.

C. Ablation Studies

1) Graph generators.: Graph generators that encode the prior knowledge of human beings are the main component of our proposed method. To investigate the influence of different graph generators, we make variants of the backbone network. The variants are divided into two categories. One uses a random graph to generate the architecture of the backbone network by fusion the input features from different views with early fusion. The other first generate the architecture of each view and then use a fusion component to exchange information. As presented in section II-B, we have three kinds of random graph generators, that is, ER model, BA model, and WS model. We set the model with 8 nodes. And the probability of ER model and WS model is set to 0.75. The connectivity of the BA model is set to 3. Meanwhile, the second category includes two variants, that is Random and ST-RWNet. We set the number of nodes for each view to 4. And the Random methods randomly select the nodes of intermediate level features from both sides. We also conduct experiments with BikeNYC and TaxiGY datasets. The result is shown in Fig. 10.

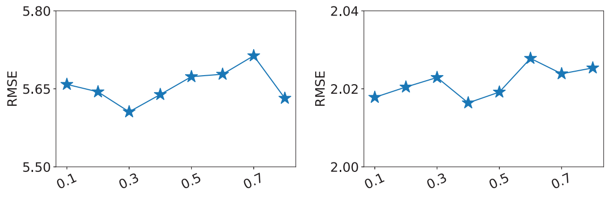
TABLE II
EXPERIMENTAL RESULTS ON SPATIO-TEMPORAL FORECASTING TASKS

	BikeNYC		TaxiGY		TaxiGN		CrowdBJ	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
`HA`	14.80	4.50	2.46	0.98	6.06	2.50	180.9	60.1
`GBRT`	7.71 ± 0.04	2.69 ± 0.01	2.257 ± 0.005	0.943 ± 0.001	4.93 ± 0.01	2.27 ± 0.00	67.4 ± 0.3	33.9 ± 0.1
`ST-ResNet`	5.79 ± 0.10	2.53 ± 0.05	$\mathbf{2.000 \pm 0.007}$	0.908 ± 0.009	$\mathbf{4.35 \pm 0.03}$	2.19 ± 0.01	53.7 ± 3.9	31.3 ± 2.2
`DeepSTN`	5.99 ± 0.04	2.93 ± 0.06	2.054 ± 0.009	0.983 ± 0.021	4.54 ± 0.01	2.38 ± 0.01	51.7 ± 3.7	34.0 ± 2.1
`CentralNet`	5.73 ± 0.04	2.69 ± 0.04	2.036 ± 0.005	0.949 ± 0.008	4.37 ± 0.01	2.22 ± 0.01	52.9 ± 3.0	32.5 ± 1.8
`ConvLSTM`	5.87 ± 0.09	2.51 ± 0.08	2.026 ± 0.007	$\mathbf{0.900 \pm 0.009}$	4.42 ± 0.01	2.17 ± 0.01	56.5 ± 2.0	33.1 ± 1.1
`ST3DNet`	5.63 ± 0.11	2.48 ± 0.05	2.011 ± 0.008	0.904 ± 0.009	4.40 ± 0.02	$\mathbf{2.16 \pm 0.02}$	53.4 ± 1.8	31.5 ± 1.4
`EPNAS`	5.70 ± 0.05	2.83 ± 0.04	2.027 ± 0.035	0.945 ± 0.017	4.43 ± 0.07	2.24 ± 0.03	52.0 ± 4.6	32.8 ± 0.8
`Darts`	5.64 ± 0.08	2.48 ± 0.03	2.028 ± 0.010	0.0933 ± 0.006	4.41 ± 0.17	2.24 ± 0.06	53.6 ± 1.7	31.5 ± 0.9
`RandomNAS`	5.62 ± 0.11	2.54 ± 0.11	2.012 ± 0.017	0.949 ± 0.024	4.58 ± 0.07	2.33 ± 0.06	55.2 ± 2.4	32.8 ± 1.3
`AutoST`	5.62 ± 0.07	2.53 ± 0.03	2.001 ± 0.010	0.935 ± 0.007	$\mathbf{4.35 \pm 0.01}$	2.17 ± 0.02	51.1 ± 1.0	31.5 ± 1.0
`ST-RWNet`	$\mathbf{5.61 \pm 0.01}$	$\mathbf{2.48 \pm 0.01}$	2.022 ± 0.008	0.953 ± 0.004	$\mathbf{4.35 \pm 0.01}$	2.19 ± 0.01	$\mathbf{50.0 \pm 1.6}$	$\mathbf{30.1 \pm 1.0}$



(a) Graph generators on BikeNYC (b) Graph generators on TaxiGY

Fig. 10. Computation time and performance comparison.



(a) Graph damage on BikeNYC (b) Graph damage on TaxiGY

Fig. 11. Graph damage.

One can observe that ER and BA models which fuse traffic features with early fusion and utilize a random graph generator perform worst. And the proposed ST-RWNet performs better than Random, which demonstrates the effectiveness of bipartite graph generators. It also shows the fusion of the intermediate level features has the potential to improve the performance of spatio-temporal traffic forecasting.

2) Graph damage.: To dig into the architecture of ST-RWNet and investigate which component dominates the traffic forecasting, we conduct experiments on graph damage by randomly removing interaction edges, which take two nodes from the different node sets. Specifically, we first sample the bipartite graph with 8 nodes, then we gradually increase the probability of drop proportion from 0.1 to 0.8 by a step size of 0.1. We retrain the model from scratch after we have removed edges at the specified probability.

As shown in Fig. 11, the results suggest there is some redundancy when the number of nodes equals 8. When drop edges with a probability between 0.3 to 0.4, we get a slight performance improvement. How to reduce redundancy is left to our future work.

D. Evaluation of Framework Settings

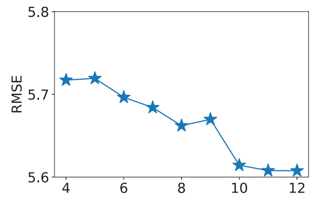
There are several hyperparameters of our proposed ST-RWNet, including, the number of nodes of the random graph, the probability of the connection of each edge, the number of channels to learn spatio-temporal patterns, and the batch size to optimize the model. We explore the way how each hyperparameter influences the performance of ST-RWNet. All experiments are conducted on the BikeNYC dataset. Fig. 12 illustrates the results.

To measure the influence of the number of nodes of the backbone network, we conduct experiments with different nodes from 4 to 12. From Fig. 12(a) we note that with the increase in the number of nodes, the performance has a certain improvement. The reason is that when the number of nodes is small, the capacity of the model is insufficient. However, when the number of nodes is greater than 10, the performance improvement is weak.

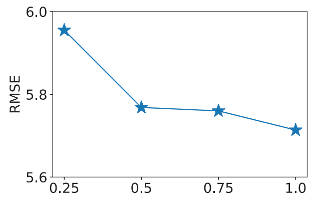
We next evaluate the influence of the probability utilized to sample the bipartite graph. Generally, the probability controls the density of the connection of the graph. If the probability is higher, the bipartite graph will be denser. On the contrary, the bipartite graph will be sparsity if the probability is close to 0. We perform experiments with a probability range from 0.25 to 1.0 with a step size of 0.25. Fig. 12(b) plots the results. One can observe that when the probability is less than 0.5, the expressive power of the model is deficient.

E. Case Study

In this section, we give some topology instances generated by bipartite graph generators as shown in Fig. 13. BG indicates the bipartite random graph instance, and the network indicates the topology of neural network architecture. Specifically, for a BG with $2N$ nodes labeled from 0 to $2N-1$, the nodes are divided into two groups, even and odd groups. Each group represents the topology of neural architecture for one



(a) Performance on number of nodes



(b) Performance on probability

Fig. 12. Studies on hyperparameters.

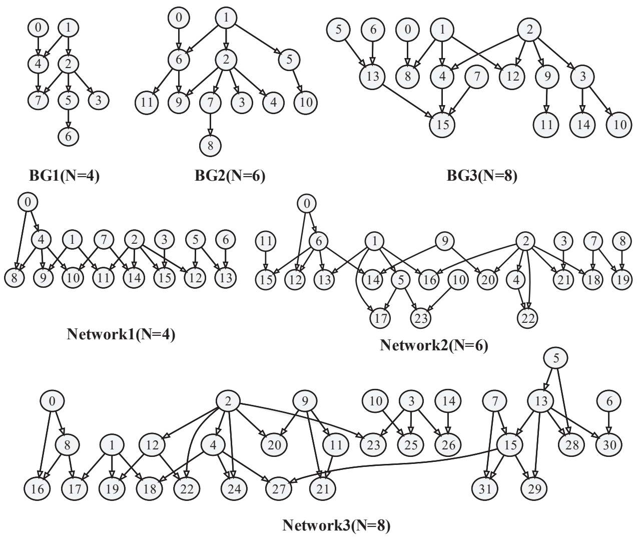


Fig. 13. Bipartite random graphs.

view. The edges between even and odd groups are split to fuse intermediate-level features. For the network, nodes with labels greater than $2N$ represent the fusion operations. One can observe that the attributes of graphs, *e.g., the number of pathways, the length distribution of pathways.* are more flexible than human designed networks.

V. Conclusion

In this paper, we propose a novel traffic forecasting method termed spatio-temporal randomly wired neural networks (ST-RWNet). To alleviate the dependencies on human experts and reduce the computation cost of existing NAS methods, we leverage spatio and temporal randomly wired neural network to construct the architecture of the backbone network. Random graphs sampled from graph generators are transferred into randomly wired neural networks to capture spatio-temporal dependencies. We conduct experiments on four real-world traffic flow forecasting datasets. The results demonstrate that ST-RWNet is compared with the state-of-the-art experts' designed architectures and NAS methods. Further, ST-RWNet reduces the computation cost with a substantial margin compared to existing NAS methods.

References

[1] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.

[2] Z. Lin, J. Feng, Z. Lu, Y. Li, and D. Jin, "Deepstn+: Context-aware spatial-temporal neural network for crowd flow prediction in metropolis," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1020–1027.

[3] T. Li, J. Zhang, K. Bao, Y. Liang, Y. Li, and Y. Zheng, "AutoST: Efficient Neural Architecture Search for Spatio-Temporal Prediction," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020, pp. 794–802.

[4] Z. Pan, S. Ke, X. Yang, Y. Liang, Y. Yu, J. Zhang, and Y. Zheng, "Autostg: Neural architecture search for predictions of spatio-temporal graph," in *Proceedings of the Web Conference 2021*, 2021, pp. 1846–1855.

[5] Y. Zhou, P. Wang, S. Arik, H. Yu, S. Zawad, F. Yan, and D. Greg, "EPNAS: Efficient Progressive Neural Architecture Search," in *Proceedings of the 2019 30th British Machine Vision Conference (BMVC 2019)*, 2019.

[6] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the Search Phase of Neural Architecture Search," *arXiv preprint arXiv:1902.08142*, 2019. [Online]. Available: http://arxiv.org/abs/1902.08142

[7] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2019-Octob, 2019, pp. 1284–1293.

[8] P. Erdos, A. Rényi *et al.*, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[9] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[10] S. Milgram, "The small world problem," *Psychology today*, vol. 2, no. 1, pp. 60–67, 1967.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[12] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[13] T. Zhou, J. Ren, M. Medo, and Y.-C. Zhang, "Bipartite network projection and personal recommendation," *Physical review E*, vol. 76, no. 4, p. 46115, 2007.

[14] Z. Liu, Y. Ma, Y. Ouyang, and Z. Xiong, "Contrastive Learning for Recommender System," *arXiv preprint arXiv:2101.01317*, 2021. [Online]. Available: http://arxiv.org/abs/2101.01317

[15] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[16] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems*, vol. 28, 2015.

[17] S. Guo, Y. Lin, S. Li, Z. Chen, and H. Wan, "Deep spatial–temporal 3d convolutional neural networks for traffic data forecasting," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 10, pp. 3913–3926, 2019.

[18] V. Vielzeuf, A. Lechervy, S. Pateux, and F. Jurie, "Centralnet: a multilayer approach for multimodal fusion," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, p. 0.

[19] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX

[20] S. Alletto, S. Huang, V. Francois-Lavet, Y. Nakata, and G. Rabusseau, "RandomNet: Towards Fully Automatic Neural Architecture Design for Multimodal Learning," *arXiv preprint arXiv:2003.01181*, 2020. [Online]. Available: http://arxiv.org/abs/2003.01181

[21] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.