Traffic Matrix Prediction and Traffic Engineering Leveraging Machine Learning Techniques

by

Le Van An

Dissertation

submitted to the Department of Informatics

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI

September 2022

Abstract

The Cisco Annual Internet Report projects that there will be 5.3 billion total Internet users by 2023, up from 3.9 billion in 2018. The global fixed broadband speeds will reach 110.4 Mbps, up from 45.9 Mbps. The great demand for Internet services (e.g., video streaming, VoIP, etc.) leads to an exponential growth in the backbone network traffic and creates a huge challenge for traffic monitoring and traffic engineering (TE). In this dissertation, we focus on solving the traffic engineering problem in backbone networks by leveraging Machine Learning (ML) techniques. We consider a network system which is managed by a centralized controller. The controller has roles of collecting network states (e.g., measuring the traffic volume) and generating routing rules to steer the traffic flows. Our main objectives are to optimize the network resources and increase the Quality of Service (QoS) by minimizing the network monitoring overhead and avoiding/reducing traffic congestion.

In traffic prediction-based TE approach, we first utilize machine learning models to have accurately predicted values of future traffic demand. Then we calculate the routing rules that can adapt to the dynamic traffic and avoid congestion based on traffic prediction. However, directly applying machine learning techniques suffers from several problems. The first and second problems are related to the low prediction accuracy caused by the missing data in both training and testing phases of the ML model. The first problem is the traffic matrix prediction with partial information. To reduce traffic monitoring cost, the controller may not measure all the network information. This can degrade the performance of traffic prediction as well as the TE tasks. The second problem is the missing values in network traffic datasets. Although missing values is a common problem in machine learning, without being properly handled, it can affect the accuracy of the trained ML model. The third problem is frequent traffic rerouting. To optimize the network routing, most of the proposed TE solutions only address the optimization problem in a single snapshot. As a consequence, traffic flows have to be frequently rerouted to adapt to the dynamic change of traffic demands. This approach leads to the degradation in the overall network's QoS. Finally, when applying ML techniques, many prior studies assume that network information such as traffic matrix or link utilization is available. However, with the explosion of traffic and the expansion of the physical network size, obtaining all the network statistics imposes high monitoring overhead. In addition, a huge amount of data is required for model training/predicting processes.

In this work, we tackle the aforementioned problems for applying ML to traffic prediction and traffic engineering: (1) improving the prediction accuracy under partial traffic monitoring, (2) inferring the missing values in the network datasets, (3) suppressing the number of traffic rerouting flows, and (4) reducing the network monitoring overhead. Firstly, to improve the performance of the traffic prediction model under partial traffic monitoring, we utilize the Convolutional Long Short-Term Memory (ConvLSTM) model and introduce the data correction algorithm to correct the outputs of the prediction model. Then, we propose an algorithm to decide the set of monitored flows in the partial flow monitoring approach. By using the proposed algorithm, we not only reduce the monitoring overhead but also achieve higher prediction accuracy. Secondly, we address the missing value problem in network datasets. Based on Recurrent Neural Network (RNN) and Graph Convolutional Neural Network (GCN), we propose a novel deep learning model to efficiently learn the dynamic correlations in partially observed data and estimate the missing values. Then, we mitigate the traffic rerouting issue in the third problem by optimizing the routing over a long-time horizon. Specifically, as segment routing (SR) approach has been widely used to solve TE problems such as minimizing the maximum link utilization of a network, we introduce a routing scheme called Multi-time-step Segment Routing (MTSR) by taking the advantage of the DNN models in traffic prediction of multiple time-steps ahead. In addition, to lower the cost of network monitoring, we propose an approach that combines traffic prediction with compressive sensing techniques. Specifically, we first leverage the DNN to predict a partial future traffic matrix using a small amount of the observed traffic and then utilize the compressive sensing technique to reconstruct the whole traffic matrix.

To evaluate the performance of the proposed approaches, we conduct experiments using different datasets of backbone network traffic. Through extensive experiments, we show that our machine learning models and the proposed solutions can overcome the impact of the missing data problem, achieve near-optimal performance in network traffic routing, and significantly reduce the number of traffic rerouting.

Acknowledgments

When reaching the last milestone of my Ph.D. journey with this thesis, I would like to express my thanks to those who made this possible.

First, I would like to express my deepest appreciation to Professor JI Yusheng, for her kind support, invaluable patience, and feedback. She always convincingly guided and encouraged me to do the right thing even though the research progress got tough. I have been lucky to be a member of her research group where my research skill becomes professional. I appreciate the advice and instructions she gave me during my past five years, which have grown me up. Without her, the goal of this dissertation would not have been completed.

I also would like to express my sincere gratitude to Professor LUI C.S. John, Professor KURIMOTO Takashi, Professor FUKUDA Kensuke, Professor KANEKO Megumi, and Professor ABE Shunji, whose encouragement, insightful comments, and hard questions helped me improve the thesis a lot.

Special thanks to all my friends, especially Dr. Phi Le, Ms. Ly Dinh, Ms. Ha Dao, Ms. Hong Vuong, and Mr. Kien Pham, for the time and experiences we had together. Without you, I would not make this wonderful journey.

Lastly, I would be remiss in not mentioning my family, especially my parents, and spouse. Their belief in me has kept my spirits and motivation high during this process.

Contents

Abstract ii

Acknowledgments v

List of Figures x

List of Tables xiii

1 Introduction 1

1.1 Traffic engineering leveraging traffic prediction 1

1.2 Challenges 3

1.2.1 Problem 1: Traffic matrix prediction with partial information . 4

1.2.2 Problem 2: Missing values in the training dataset 5

1.2.3 Problem 3: Frequent traffic flows rerouting 6

1.2.4 Problem 4: High monitoring overhead 7

1.3 Objectives and contributions 7

1.4 Dissertation organization 9

2 Background and related work 11

2.1 Traffic engineering 11

2.1.1 Traffic matrix 11

2.1.2 Traditional TE 12

2.1.3 Segment Routing 14

2.2 Machine learning techniques 14

2.2.1 Long Short-Term Memory 15

2.2.2 Graph neural network-based traffic prediction 17
2.3 Compressive sensing 19
2.4 Related work 20
2.4.1 Traffic matrix prediction 20
2.4.2 Network traffic imputation 21
2.4.3 Traffic engineering with segment routing 23

3 Traffic matrix prediction with partial information **25**
3.1 Problem statement 26
3.2 The low accuracy of traffic matrix prediction with partial information . 27
3.3 ConvLSTM-based traffic matrix prediction 29
3.3.1 Overview of the proposed approach 29
3.3.2 Future traffic matrix prediction 30
3.3.3 Backward data correction 31
3.3.4 The skip connection 33
3.3.5 The backward data correction algorithm 33
3.4 Monitored flow determination 35
3.5 Performance evaluation 37
3.5.1 Evaluation metrics 37
3.5.2 Dataset 37
3.5.3 Model setting 38
3.6 Experimental results 38
3.6.1 Comparison of LSTM and ConvLSTM network-based traffic matrix prediction 39
3.6.2 Evaluating the data correction module 40
3.6.3 Evaluating of the skip connection module 41
3.6.4 Evaluating the monitored flow determination module 43
3.6.5 Evaluating the performance of one-step-ahead prediction . . . 44
3.6.6 Evaluating the performance of multi-step-ahead prediction . . 45
3.6.7 Evaluating the effect of the input size (H) 47
3.6.8 Evaluating the time complexity 48
3.7 Summary 49

4 Network traffic imputation 50
4.1 Problem statement . . . 50
4.2 Graph convolutional recurrent neural network for imputing network traffic . . . 51
4.2.1 Input layer . . . 52
4.2.2 Temporal feature learning with BiLSTM . . . 53
4.2.3 Spatial feature learning with GCN . . . 54
4.3 Performance evaluation . . . 55
4.3.1 Datasets and baseline methods . . . 55
4.3.2 Performance metrics . . . 55
4.3.3 Generating synthetic missing data . . . 55
4.3.4 Evaluating the impacts of the input size . . . 56
4.3.5 Evaluating the imputation accuracy on random missing . . . 57
4.3.6 Evaluating the imputation accuracy on missing data caused by link failures . . . 58
4.3.7 Evaluating the impacts of imputed datasets on traffic prediction-based TE . . . 61
4.4 Summary . . . 63

5 Traffic prediction-based traffic engineering 64
5.1 Problem statement . . . 65
5.2 Multi-time-step Segment Routing . . . 67
5.2.1 Problem formulations . . . 67
5.2.2 Theoretical analysis . . . 69
5.2.3 Traffic prediction for multi-time-step segment routing . . . 72
5.3 Local search algorithm for solving MTSR problem . . . 73
5.3.1 Search space reduction techniques . . . 73
5.3.2 Link and flow selection strategy . . . 75
5.3.3 Local search for multi-time-step segment routing . . . 75
5.4 Partial traffic prediction and compressive sensing . . . 76
5.4.1 Compressive sensing-based network traffic reconstruction . . . 77
5.4.2 Reconstruct the traffic matrix from the partial traffic prediction 78
5.4.3 Training phase . . . 80

5.4.4 Testing phase 82
5.5 Performance evaluation 83
5.5.1 Datasets and performance metrics 83
5.5.2 Evaluating the MTSR approach without traffic prediction . . . 83
5.5.3 Evaluating the MTSR approach with traffic prediction 85
5.5.4 Evaluating the scalability of the LS2SR algorithm 88
5.5.5 Evaluating the MTSR approach with compressive sensing . . . 90
5.6 Summary 91

6 Conclusion **93**
6.1 Summary of the dissertation 93
6.2 Discussion and open issues 94
6.2.1 Network traffic datasets 94
6.2.2 Explainable ML/DNN techniques 95
6.2.3 Traffic prediction accuracy versus network routing performance 96
6.3 Future direction 96

Bibliography **99**

List of Publications **107**

List of Figures

1.1 ML-based traffic engineering in SDN network. 2
1.2 The predicted value is used as input for the next prediction step since the traffic is not measured. 4
1.3 Imputation process is done before training the prediction model. 6
1.4 Outline of the dissertation. 9

2.1 An example of a traffic matrix. 12
2.2 Example of segment routing and SR policy. 14
2.3 (a) The architecture of LSTM unit and (b) LSTM network. 16
2.4 The structure of the Bidirectional LSTM network. 16
2.5 Encoding-forecasting ConvLSTM network for precipitation nowcasting [65]. 17
2.6 2-D convolution versus graph convolution [58]. 18
2.7 Example of network link load traffic under block missing scenarios on two different days for three links in the Abilene network. The dot lines represent the missing blocks, while the stars indicate the observed data. 23

3.1 The effect of monitoring ratio on prediction accuracy (One-step-ahead prediction using the LSTM network). 28
3.2 The overall system of traffic matrix prediction with partial information. 29
3.3 The sequence of H input matrices of ConvLSTM network. 30
3.4 The structure of ConvLSTM network for traffic matrix prediction. . . . 31
3.5 The forward and backward networks with skip connection. 32
3.6 Example of data correction. 35

3.7 The comparison between LSTM and ConvLSTM-based approach. 40
3.8 The pairwise correlation of 25 traffic flows in the 5×5 matrix. 41
3.9 The effect of data correction module. 42
3.10 Comparison FWBW-LSTM model with and without skip connection. . 42
3.11 The effect of monitoring strategies on traffic matrix prediction. 43
3.12 Performance comparison in one-step-ahead prediction. 44
3.13 Performance comparison in multi-step-ahead prediction ($T = 3$). 46
3.14 The Error Ratio of all models with different multi-step-ahead prediction. 46
3.15 The effect of input size. 48

4.1 The architecture of GCRINT model. 52
4.2 The design of the input layer. 53
4.3 Example of random and block missing scenarios of Abilene dataset in 144 time-steps (12 hours). The black dots represent the missing values. 56
4.4 GCRINT on different input sizes. 57
4.5 The comparison of all methods in terms of MAE with different missing scenarios. 58
4.6 The cumulative distribution of times between failures over the entire network (a) and the mean time between failures per link (b). 59
4.7 Example of missing values caused by link failures of Abilene dataset in 144 time-steps (12 hours). The black dots represent the missing values. 60
4.8 Traffic Engineering leveraging network traffic imputation. 61
4.9 The comparison of imputation methods in traffic prediction-based TE. . 63

5.1 Illustration of 2-segment routing [5]. 66
5.2 Examples of redundant paths in 2-segment routing. 74
5.3 Partial traffic measurement. The traffic matrix X is represented as a vector. 78
5.4 Illustration of partial traffic prediction and matrix reconstruction using compressive sensing. 79
5.5 Generating the training dataset from network traffic matrices. 80
5.6 The maximum link utilization with different routing approaches. 84
5.7 The number of routing changes of different routing approaches. 84
5.8 The maximum link utilization of the first 100 time-steps. 85

5.9 The maximum link utilization of different routing approaches. 86
5.10 The average number of rerouting flows per time-step. 87
5.11 The maximum link utilization of different groups of network topology. 88
5.12 The number of rerouting flows per time-step of different groups of network topology. 89
5.13 The average delay per time-step of different groups of network topology. 90
5.14 The MLU of MTSR-CS with different percentages of monitored flows. . 90

List of Tables

2.1 Summary of deep learning models for traffic prediction. 15

3.1 The configurations of LSTM and ConvLSTM networks. 38
3.2 Experiment configurations. 39
3.3 The results of ARIMA model in one-step-ahead prediction. 44
3.4 The results of ARIMA model in multi-step-ahead prediction. 45
3.5 The training and prediction time of LSTM and FWBW-LSTM. 48
3.6 Comparison in the number of parameters and the amount of used memory. 49

4.1 The missing rate of the generated datasets. 60
4.2 Performance comparison of imputation methods on recovering missing values in network traffic dataset caused by link failure events. 61

5.1 The differences of the problem formulations. 67
5.2 Details of the real traffic datasets. 83

1 Introduction

1.1 Traffic engineering leveraging traffic prediction

Internet traffic engineering, or TE for short, is concerned with the performance optimization of operational networks. A major goal of traffic engineering is to facilitate efficient and reliable network operations while simultaneously optimizing network resource utilization and traffic performance [32]. The TE's goals can be done by steering the traffic in the network to avoid the congestion, protect the links/nodes failures, and achieve the customer agreements (e.g., bandwidth guarantee). Due to its important roles, TE has attracted massive attention from both academia and industry throughout the history of Internet development. Especially, in recent years, we have witnessed the exponential growth in Internet traffic due to the increasing number of mobile/IoT devices along with the shifting in the user behaviors due to the pandemic [11, 43] (e.g., working/entertaining from home). The increase in the traffic volume creates a strong urge to have more efficient TE solutions that can adapt to the rapid changes in network traffic.

Recently, machine learning (ML), especially deep neural network (DNN), has achieved a lot of success in solving and/or improving the performance of many networking tasks. Traffic engineering is also benefited from the successes of the modern ML/DNN techniques to become promising approaches compared to existing solutions. The ML/DNN techniques can either be used to predict future traffic demands or directly generate the routing rules. For example, network traffic routing benefits from several ML/DNN models (e.g., Long Short-Term Memory network) to have accurately predicted values of future traffic demand, and therefore, the routing rules can be calculated to adapt to the dynamic network traffic and avoid the congestion. The experiment's results demonstrate that applying ML/DNN in solving TE problems is a promising approach.

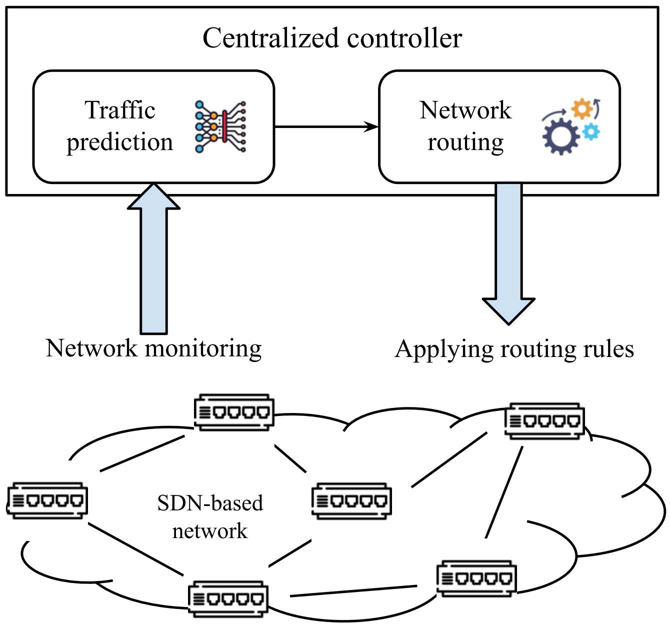


Figure 1.1: ML-based traffic engineering in SDN network.

In this dissertation, we focus on solving the traffic engineering problem in the backbone network by leveraging machine learning techniques. We consider a network system managed by a centralized controller. The decoupling of the control plane and the data plane in the network architecture such as Software Defined Networking (SDN) has opened numerous potentials for applying ML techniques in traffic engineering. Being able to provide the global view of real-time network's state and traffic flow information, the SDN controller becomes suitable and efficient to utilize/execute the advanced ML techniques for network routing. Many studies have proposed ML-based solutions for routing optimization problems. Compared to traditional approaches, ML-based solutions have some advantages. ML algorithms can quickly generate routing rules after being trained. In addition, ML algorithms do not need an exact mathematical model of the underlying network [60].

Figure 1.1 illustrates an ML-based network routing approach in SDN-based network. ML techniques are used to estimate future traffic demands based on the monitored network information. The centralized controller is responsible for collecting/monitoring the network information such as traffic matrices and calculating the routing policy for each traffic flow using the predicted traffic demands. Thanks to the outputs of traffic prediction, the network controller can calculate the proactive routing rules in advance to adapt to the dynamic traffic in the near future. In this way, the controller can take appropriate actions before traffic congestion occurs and improve Quality of Service (QoS). Therefore, by improving the prediction accuracy, we can achieve better performance in traffic engineering.

However, directly applying ML in traffic prediction and network routing faces several practical problems related to missing data in the network measurement, frequent traffic rerouting, and high monitoring overhead. The details of the problems will be presented in Section 1.2.

1.2 Challenges

In the past few years, machine learning and deep neural networks have achieved great success in many research areas as well as real-world problems such as computer vision, natural language processing. In networking field, ML/DNN techniques are also believed to be the keys for breaking the performance of traditional methods used in many networking problems. However, when applying ML techniques, especially in traffic prediction-based TE, we face several problems. The details of the problems will be described as follows.

1.2.1 Problem 1: Traffic matrix prediction with partial information

We first consider the problem of traffic matrix prediction with partial information. Traffic matrix contains the monitored traffic volume of all source-destination flows in the network. It plays an important role in many networking applications such as anomaly detection and network routing. Accurately estimating the future traffic matrices helps to improve the adaptability of the routing mechanism in a dynamic network. In traffic matrix prediction problem, although many studies have applied ML/DNN techniques, the existing solutions focus on improving the accuracy of the prediction model and assume that network measurements such as traffic matrices are available. However, due to high monitoring overhead, the traffic matrix may not be fully obtained (e.g., by applying sparse monitoring [61]). We assume that the network controller can decide the subset of flows that will be measured. This decision is made by the controller after every time-step. In this scenario, we face two problems: the poor prediction accuracy due to the imprecise values in the input and how to determine which flows to be measured in the next time-step.

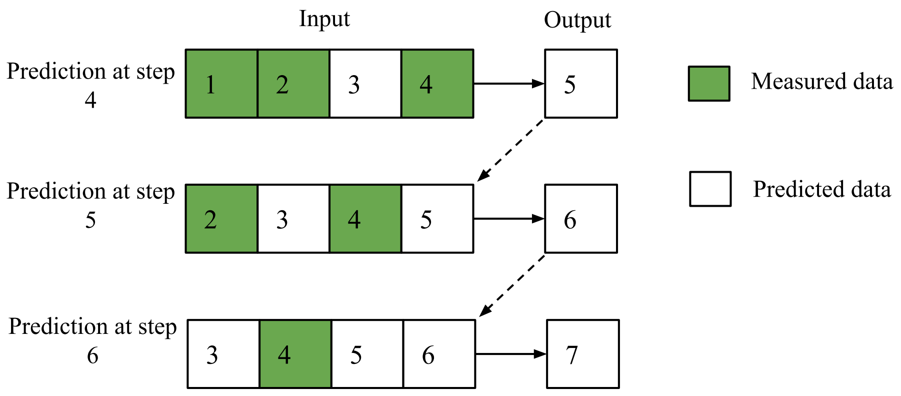


Figure 1.2: The predicted value is used as input for the next prediction step since the traffic is not measured.

In the first problem, at every prediction step, the missing entries in the traffic matrix will be replaced by the predicted values of the previous prediction steps. Figure 1.2 shows the example of predicting future traffic of one flow by using predicted values as input when the actual data is not measured. At time-step 5, the controller does not have the measured data, the predicted value (i.e., the output of step 4) is used as input. When performing traffic prediction at step 5, since the input sequence contains some imprecise data, the prediction accuracy will be decreased. Therefore, the low monitoring ratio leads to a higher number of imprecise values in the input sequence, hence the higher prediction error. As the error in the input data results in the error in the prediction's output, we call this problem the accumulative error.

In the second problem, the controller needs to determine which flows to be measured at the next time-step. To reduce the monitoring overhead, we do not measure all traffic flows, thus choosing appropriate flows to monitor in each time-step becomes a critical factor in reducing the prediction error. One of the common approaches in choosing a monitored flow set is to maintain fairness among all the flows. Specifically, the gaps between every two consecutive monitored time-steps of every flow are kept to be approximately the same. However, this method may not be effective since the network flows are dynamic and have various temporal fluctuation patterns. To deal with this problem, we propose a novel scheme for selecting the next monitored flows, which exploits the previous prediction results.

1.2.2 Problem 2: Missing values in the training dataset

When applying the machine learning, the quality of the training dataset has a huge impact on the performance of the trained model. The poor data quality results in low prediction accuracy. In the networking field, the network dataset usually contains missing data due to unexpected accidents such as a link down and packet loss. The missing data problem can prevent advanced analysis and downgrades downstream applications such as traffic engineering and anomaly detection.

We consider the problem when training the traffic prediction model with data that contains missing values. In the traffic prediction-based TE approach, a deep neural network is trained to estimate future traffic demands. In most existing traffic prediction studies, the model training phase is usually omitted or briefly mentioned, especially the data preprocessing step. Several problems will be handled before the data is used in the model's training step. One of the most common problems in the data preprocessing is the missing values. Missing values appear in most multivariate time series, especially in the monitored network traffic data due to high measurement cost and unavoidable loss. Figure 1.3 illustrate the data preprocessing step in handling

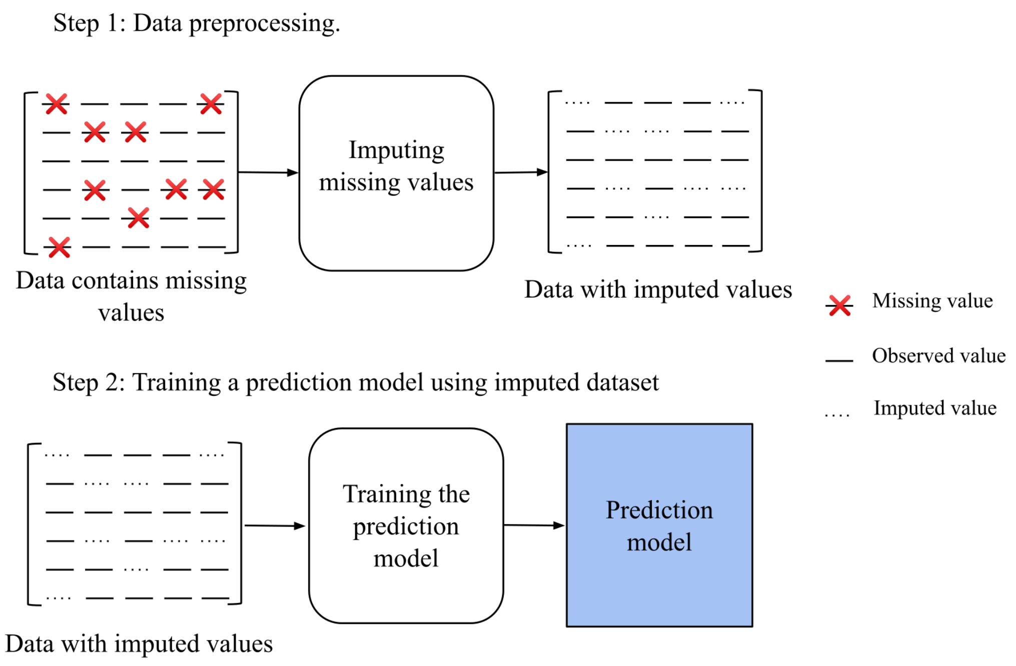

Step 1: Data preprocessing.
Data contains missing values
Imputing missing values
Data with imputed values
Missing value
Observed value
Imputed value
Step 2: Training a prediction model using imputed dataset
Data with imputed values
Training the prediction model
Prediction model

Figure 1.3: Imputation process is done before training the prediction model.

the missing values. All the missing entries in the data need to be imputed before the training step. Training the deep learning model with missing data or low accuracy of imputed data will degrade the model's performance.

1.2.3 Problem 3: Frequent traffic flows rerouting

Changing the routing path of traffic flows is the common method in TE to avoid traffic congestion. Most of the proposed TE solutions only address the optimization problem in a single snapshot (which is called a "time-step" in this dissertation) without considering the long time horizon. These approaches usually use a traffic matrix [54] (i.e., traffic demands) of the corresponding time-step as the input to solve the optimization problem and obtain the routing rules. However, due to the dynamic network behavior, traffic matrix often varies over time, which may lead to the high link utilization and network congestion. To adapt to traffic demand variation, the traditional TE solutions may need to reroute many flows to balance the traffic loads, leading to significant network disturbance and service disruption. Depending on the traffic fluctuation, network optimization and traffic rerouting can be performed with high frequency (e.g., at every minute). In [9], the authors evaluate the impact of frequent route changes on TCP flows. Due to the transparency between flow rerouting and congestion control algorithm, the total throughput of the network is sometimes observed to nearly 50% drop after rerouting a large number of flows. In addition, changing the path of traffic flows frequently may degrade the QoS due to the packet loss or out-of-order delivery as well as the increase in the flow's completion time [69], [68]. Therefore, reducing the number of traffic rerouting flows over a long time horizon need to be taken into the consideration in designing the TE solutions.

1.2.4 Problem 4: High monitoring overhead

When applying ML techniques in traffic engineering, many prior works only focus on solving routing problems and assuming the network statistics such as traffic matrix or link utilization are available. However, with the explosion of traffic and the expansion of the physical network, obtaining all the network statistics imposes high monitoring overhead. In addition, applying ML/DNN into networking also needs a huge amount of monitored data for training/predicting processes. Although the quality of the measurements may have a huge impact on the performance of the TE solution [27], there are only a few studies that consider the joint problem of network monitoring and traffic engineering. Moreover, the scalability issue of applying ML/DNN techniques is often omitted in many ML-based TE studies.

1.3 Objectives and contributions

In this dissertation, our objective is to develop a traffic prediction-based traffic engineering approach that utilizes machine learning techniques. We study traffic engineering in the backbone network and tackle the aforementioned problems (Section 1.2): predicting traffic matrix with partial information, imputing missing values in network datasets, and reducing the number of traffic rerouting as well as the monitoring overhead.

We introduce a mechanism for correcting the imprecise entries in traffic data, which results in the improvement in the accuracy of the traffic matrix prediction task. Then, we propose a novel deep learning model for inferring the missing values in the network datasets by exploiting the spatio-temporal in traffic data. Next, we mitigate the traffic rerouting by optimizing the routing over a long time horizon. We introduce a routing scheme called multi-time-step segment routing (MTSR for short) by taking the advantages of the Deep Neural Network in traffic prediction of multiple time-steps ahead. Finally, to lower the cost of network monitoring, we propose an approach that combines DNN-based partial prediction with the compressive sensing technique.

The main contributions are summarized as follows:

- We propose a prediction scheme based on Convolutional Recurrent Neural Network model which can iteratively improve the predicted results of the previous prediction. Since the predicted results is latter used for the next prediction round, this scheme can improve the prediction accuracy in overall (**problem 1**).
- To alleviate the impact the missing data problem, we design a novel DNN model based on the Graph Neural Network for imputing the missing values (**problem 2**).
- We address the routing path change issue in the traffic engineering by formulating the multi-time-step segment routing (MTSR) problem as an Integer Linear Programming problem. MTSR leverages the advanced DNN model for multiple time-steps traffic prediction in order to reduce the number of rerouting flows over a long time horizon (**problem 3**).
- Considering the impact of the prediction accuracy, we design three versions of the MTSR problem and provide the theoretical analysis on the performance of these proposed versions. In addition, we introduce an efficient local search based algorithm called LS2SR to solve the MTSR problem in the context of time-constrained optimization (**problem 3**).
- We reduce the high monitoring cost in traffic engineering by proposing an approach that combines the partial network traffic prediction using Deep Neural Network model with compressive sensing technique (**problem 4**).
- We conduct extensive experiments using different real backbone network datasets to evaluate the performance of our proposed methods and compare them to the

state-of-the-art approaches.

1.4 Dissertation organization

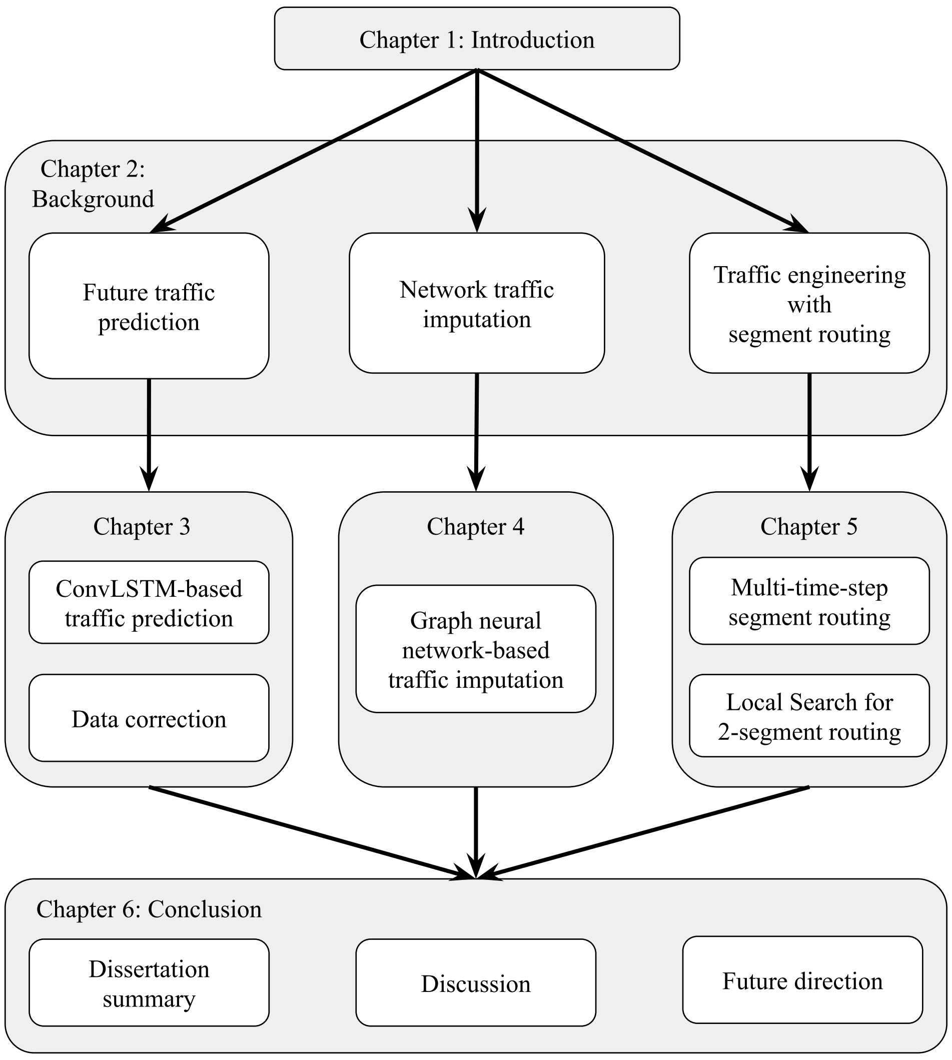

Chapter 1: Introduction
Chapter 2: Background
Future traffic prediction
Network traffic imputation
Traffic engineering with segment routing
Chapter 3
ConvLSTM-based traffic prediction
Data correction
Chapter 4
Graph neural network-based traffic imputation
Chapter 5
Multi-time-step segment routing
Local Search for 2-segment routing
Chapter 6: Conclusion
Dissertation summary
Discussion
Future direction

Figure 1.4: Outline of the dissertation.

The remainder of this dissertation is organized as follows. Chapter 2 elaborates on the background and related work in the area of traffic engineering and machine learning, respectively. We present our proposed DNN model for future traffic prediction with partial information in Chapter 3. Then, Chapter 4 gives the details of our approaches to tackle the missing values problem in the network data. In chapter 5, we introduce the routing scheme called multi-time-step segment routing (MTSR) to reduce the number of rerouted traffic flow over the long-time horizon. First, we present the mathematical formulations of the MTSR and the theoretical analysis. Then, we describe an effective heuristic algorithm called LS2SR to solve the MTSR problem. We also give the details of our proposed method for reducing network monitoring cost by applying partial traffic prediction and compressive sensing techniques. Finally, in chapter 6, we summarize the contributions of this dissertation and discuss the limitations and the future direction. Figure 1.4 illustrates the outline of our dissertation.

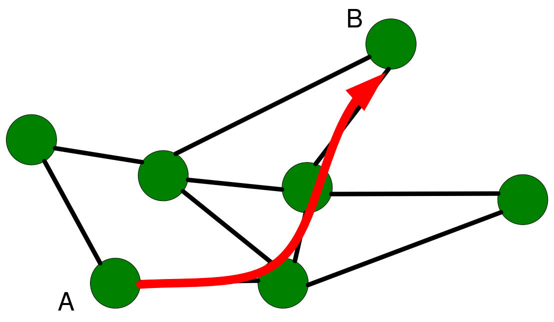
2 Background and related work

In this chapter, we introduce the basic background and notations that are used throughout the dissertation. First, we provide a brief overview of traffic engineering and the current state-of-the-art TE approaches in Section 2.1. Then, we introduce the ML/DNN as well as the compressive sensing techniques in Section 2.2 and 2.3. Finally, in Section 2.4, we present the existing solutions in traffic prediction, missing data imputation, and traffic engineering.

2.1 Traffic engineering

2.1.1 Traffic matrix

According to [54], a traffic matrix is defined as an abstract representation of the traffic volume following between a set of source and destination pairs. Traffic matrix is a $N \times N$ matrix (N is the number of nodes in the network) where each element denotes the amount of aggregated traffic from a source to a destination. The source



$$\begin{pmatrix} X_{AA} & X_{AB} & X_{AC} & \cdots \\ X_{BA} & X_{BB} & X_{BC} & \cdots \\ X_{CA} & X_{CB} & X_{CC} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Figure 2.1: An example of a traffic matrix.

and destination, depending on the network layer of study, could be routers or the whole network. In this work, we focus on the backbone network where the sources and destinations are the nodes (i.e, routers) in the network and each element of the traffic matrix depicts the number in bytes of the aggregated flow between two nodes. Figure 2.1 shows an example of the traffic matrix given by the topology on the left. Logically, the values of $X_{AA}, X_{BB}, \ldots$ are zero, however, in most cases, they are non-zero because of aggregation of devices such as a PoP or an AS.

Playing a role as a snapshot of the network situation, the traffic matrix is utilized in a variety of network management tasks and applications, such as network optimization, protocol design and anomaly detection. We can exploit a set of traffic matrices for offline tasks such as network analysis, finding unexpected events (e.g., malicious, DDoS) or capacity planning. Besides that, if the traffic matrix is exactly known (by doing network monitoring) or can be predicted precisely, along with the topology information, the network operator can perform traffic routing or load balancing effectively and efficiently.

2.1.2 Traditional TE

Congestion minimization is one of the most important objectives in traffic engineering. This objective is often achieved by minimizing the maximum link utilization (MLU) of the network. The MLU is calculated as follows:

$$u = \max_{e \in E} \frac{load(e)}{c(e)} \tag{2.1}$$

where E, $load(e)$, and $c(e)$ are the a set of all link in the network, total traffic load on link e, and the capacity of link e, respectively. The traffic load on link e is calculated by using the traffic matrix and the routing rules.

In general, there are two routing strategies: adaptive routing and oblivious routing. In adaptive routing, a path is chosen based on current network information such as the traffic flow's demand. Therefore, it can dynamically adapt its routing decisions to the current traffic in the system. In theory, most of the works in adaptive routing rely on solving the Multi-Commodity Flow (MCF) problem, obtaining the fractional solutions in which the traffic flows are arbitrarily split and routed through different paths. In practice, due to the simplicity, shortest path routing-based approaches such as OSPF, and IS-IS are widely deployed in many ISP networks. By tuning the link weights in a distributed manner, shortest path routing can compute the near-optimal set of forwarding paths. However, their drawbacks are the high re-convergence time and poor performance when there are changes in the network topology or the traffic demands. Later, routing techniques such as MPLS and RSVP have provided flexibility and increased the TE performance by supporting explicit routing paths. However, MPLS-TE solutions are known for their long convergence times and high cost of maintaining the TE tunnels.

Different from the adaptive routing mentioned above, oblivious routing performs traffic routing independently with the current traffic in the network [3], [40]. In fact, in oblivious routing, the path from a source node to a destination node may be chosen by only using the information of the network topology. The routing rules of the oblivious routing strategy are calculated only once time and stored at each node in the network via routing tables. Therefore, it is easier to implement and does not cause the rerouting problem. However, oblivious routing faces possible performance loss in the highly dynamic traffic system. This is the result from restricting the routing rules to a fixed set of rules instead of considering their adaptive counterparts [41].

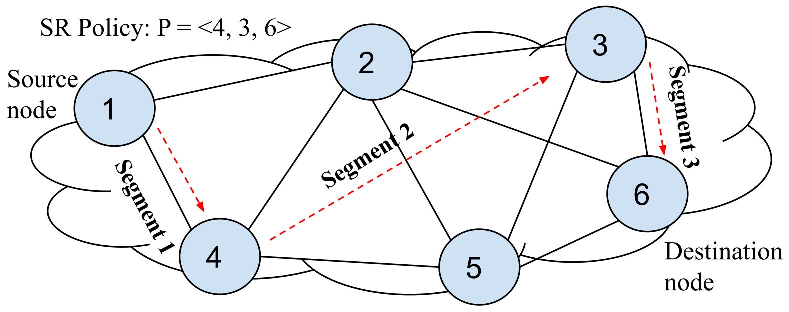


Figure 2.2: Example of segment routing and SR policy.

2.1.3 Segment Routing

Based on the source routing paradigm, segment routing (SR) allows the source node (or ingress node) to insert a list of Segment Identifiers (Segment ID or SID) to the packet header. The segment list becomes the set of instructions (SR policy or SR rule) to the network devices to steer the packet. Although the SID can be used to identify both the node or link in the network, for simplicity, we only consider the node-segment in this study. Figure 2.2 shows an example of segment and SR policy. In this example, traffic from the node 1 is routed to the node 6 using 3 segments: 1-4, 4-3, and 3-6. The SR policy is a ordered list of the node that traffic from the source node need to visit (i.e., <4, 3, 6>), including the destination node. The packets from the source node need to be routed through all the nodes in the SR policy before being forwarded to the destination. The shortest path routing (e.g., OSPF) is used to forward the packet within the segment (e.g., traffic in segment 4-3 is routed through the shortest path from node 4 to node 3).

2.2 Machine learning techniques

Recently, machine learning techniques have been applied to various networking problems such as traffic routing, demands prediction, and anomaly detection. In traffic engineering, ML can be used for estimating future traffic demands or directly generating the routing rules. In this section, we give the background knowledge about ML/DNN models used for traffic prediction problems. Table 2.1 summarizes the deep neural network models that are used for traffic prediction.

Type	Model	Model name	Characteristics
RNN	LSTM	Long Short-Term Memory	Well-known for handling the time series data Learning the temporal relations
	BiLSTM	Bidirectional LSTM	Using two LSTM networks Learning the temporal relations in both directions of the time series data
	ConvLSTM	Convoutional LSTM	Combining the LSTM and Convolutional neural network Learning the temporal and spatial relations in the traffic data
GNN	GCN	Graph Convolutional Neural Network	Learning the spatial relations of data in which the relations have graph-based structure
	GWN	Graph WaveNet	Learning both temporal and spatial relations of data using graph convolutional neural network

Table 2.1: Summary of deep learning models for traffic prediction.

2.2.1 Long Short-Term Memory

LSTM unit and LSTM network

A Long Short-Term Memory network (LSTM) is a special Recurrent Neural Network (RNN) that replaces a standard RNN unit with a LSTM unit. The major innovation of the LSTM unit is the memory cell c_t which accumulates the state information. The LSTM network has the ability to remove or add information to the cell state, carefully regulated by structures called gated. Gates are the way to optionally let information through (showed in Fig. 2.3(a)). They are composed of the non-linear network layer (i.e., sigmoid, relu, etc.). The LSTM network has been proved to be stable and powerful for modeling long-range dependencies of time series data in various problems. Therefore it is well-suited for processing and making predictions based on time series or sequence data. Indeed, LSTM has been applied in many real-life sequence modeling problems. The unfolded model of the LSTM network (Fig.2.3(b)) shows that the input is processed step-by-step and the outputs of previous steps are used as the input for the next step. This architecture along with the advantages of the memory cell in LSTM units make the LSTM network especially suitable for solving the time series forecasting.

Bidirectional LSTM network

In several problems of Natural Language Processing domain such as predicting the missing words in a sentence, we need not only the information of the words before the missing position but also the information of the words after that. The original LSTM model does not perform well in this problem since it only has one direction

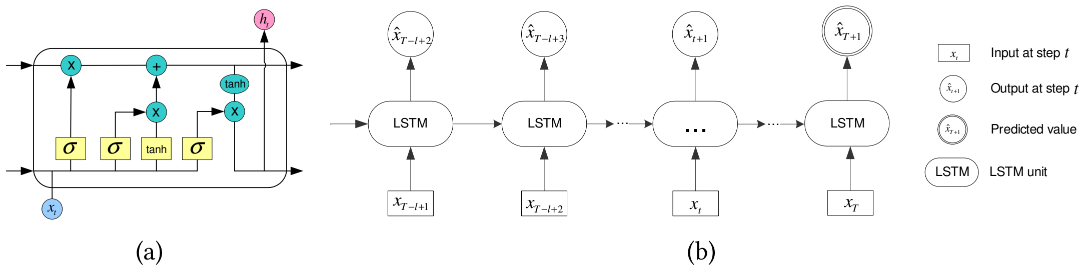


Figure 2.3: (a) The architecture of LSTM unit and (b) LSTM network.

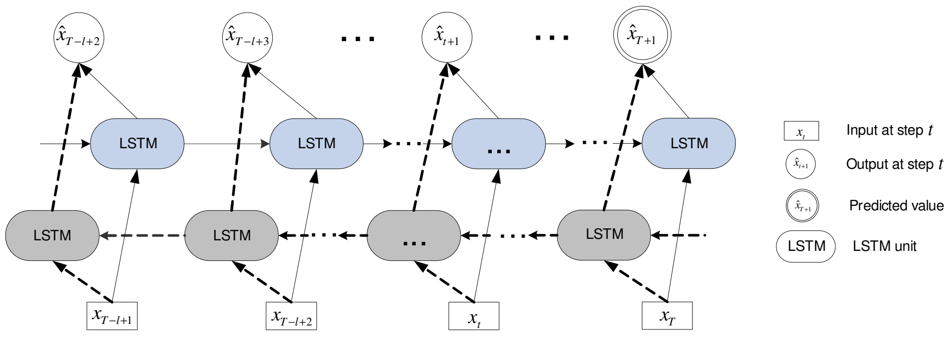


Figure 2.4: The structure of the Bidirectional LSTM network.

in the data flow. In [47], the authors proposed a modification of the LSTM model called Bidirectional LSTM. They added another LSTM network to the model and then concatenated or averaged the outputs from both LSTM networks. The second network gets the same input as the first network but in the backward order (as shown in Fig.2.4). The BiLSTM network can "learn" the data from both directions and therefore well perform in the problems that need to predict/estimate the value in the middle of time-series data.

Convolutional LSTM network

When working with the data that has a grid-based structure (e.g., image, matrix), the best approach is to use Convolutional Neural Network (CNN) to extract the spatial features. Therefore, to extract both spatial and temporal features in the sequential images/matrices, the authors in [65] have proposed the Convolutional LSTM network (ConvLSTM) that combines the convolutional operation and the LSTM network. ConvLSTM is a type of recurrent neural network like LSTM, but the internal matrix

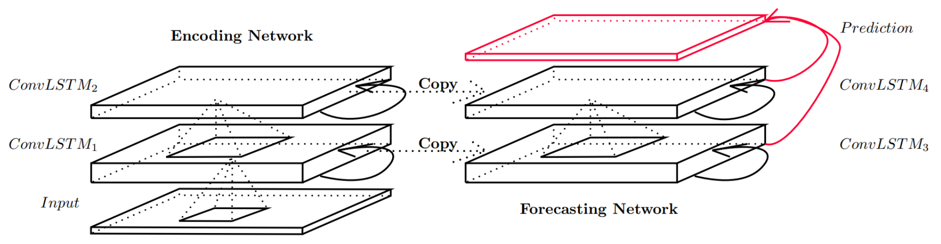


Figure 2.5: Encoding-forecasting ConvLSTM network for precipitation nowcasting [65].

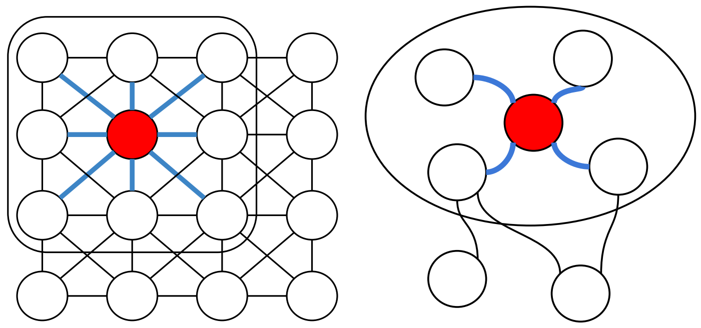
multiplications are exchanged with the convolutional operations. As result, the data that flows through the ConvLSTM has the dimension of 2D or 3D instead of being just a 1D vector. Figure 2.5 illustrates the architecture of ConvLSTM network used for precipitation nowcasting.

2.2.2 Graph neural network-based traffic prediction

Graph Neural Network

In recent years, Graph Neural Network (GNN) [59] has become one of the most successful deep learning models, showing the state-of-the-art in various real-world problems. Figure 2.6, as an example, illustrates the difference in learning the spatial information between GNN and CNN. The CNN usually works on the data that has grid-based structures such as an image or a matrix. The convolution operation takes the weighted average of node values along with its neighbors. The neighbors of a node are ordered and have a fixed size. On the other hand, GNN can represent the hidden relations of the data as a graph and take the average value of the node features along with its neighbors. Different from the image data, the neighbors of a node can be unordered and variable in size.

By exploiting the graph-based relations of the data, GNN has been used to model lots of complex problems such as social networks and brain networks. In the network traffic prediction problem, GNN can efficiently capture the spatial dependency in the traffic networks, which is naturally represented by non-Euclidean graph structures. This graph, which is called traffic graph, is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Each node in the traffic graph is associated with a



(a) 2-D convolutional operation. (b) Graph convolutional operation.

Figure 2.6: 2-D convolution versus graph convolution [58].

traffic flow in the network. For a single-time step t, the node feature matrix $X_t \in \mathbb{R}^{N \times F}$ for the traffic graph $\mathcal{G}$ contains a specific traffic matrix of the network, where N is the number of nodes in $\mathcal{G}$ and F is the number of traffic features (e.g., volume, delay, measurement indicator). An edge $e \in \mathcal{E}$ represents the relation between two nodes (or two traffic flow).

In [58], the authors have defined a graph-based traffic prediction as follow: find a function f which generate $y = f(X, \mathcal{G})$, where y is the future traffic states, $X = [X_1, ..., X_t]$ is the historical traffic state, and $\mathcal{G}$ is the traffic graph.

Graph Convolutional Network

Graph Convolutional Network (GCN) is a special type of deep neural network based on the combination between GNN and CNN. GCN is used to obtain the hidden features from the data given its graph-based structure. In the GCN-based traffic prediction approach, the spatial relation among the nodes in the traffic graph is extracted by aggregating and transforming the neighborhood information. Let $A \in \mathbf{R}^{N \times N}$ denote the adjacency matrix of $\mathcal{G}$, and $\tilde{A}$ be its normalized matrix. $Z \in \mathbb{R}^{N \times M}$ denotes the output and $W \in \mathbb{R}^{D \times M}$ denotes the parameters of the deep learning model. In [24], the graph convolutional operation is defined as $Z = \tilde{A}XW$. In [29], Li et al. proposed diffusion convolutional layer in which the aggregating neighborhood's information process is done with K steps. The diffusion process is characterized by a random walk on $\mathcal{G}$ with transition matrix $P = A/sumrow(A)$ and is expressed in Equation 2.2.

$$Z = \sum_{k=0}^{K} P^k X W \tag{2.2}$$

Graph WaveNet

One of the limitations of GCN is that it can only capture the spatial dependency on a fixed graph structure and assumes that the relations among the nodes (i.e., $\mathcal{E}$) in the graph are given. However, in many problems such as traffic prediction, the relations among the traffic flows may not be clearly defined and can also be changed over time (i.e., the dynamic relations). To overcome this limitation, Wu et al. proposed a novel deep learning model for spatial-temporal graph modeling, called Graph WaveNet (GWN) [59]. To learn the dynamic relations in the graph, a self-adaptive adjacency matrix is added in the graph diffusion process. The graph convolutional operation with self-adaptive adjacency matrix is expressed in Equation 2.3.

$$Z = \sum_{k=0}^{K} (P^k X W + \tilde{A}_{adp}^k X W_{adp}) \tag{2.3}$$

where $\tilde{A}_{adp} = SoftMax(ReLU(E_1, E_2^T))$ is the self-adaptive adjacency matrix. $\tilde{A}_{adp}$ is calculated using two learnable parameters $E_1, E_2 \in \mathbb{R}^{N \times c}$. The $ReLU$ function is used to eliminate the weak connection and the $SoftMax$ is applied to normalize the adjacency matrix. By using the learnable parameters to construct the adjacency matrix, GWN can learn the relations among the nodes in $\mathcal{G}$ from the training data.

2.3 Compressive sensing

Compressive sensing (CS) is a well-known technique in signal processing that is used for the acquisition and/or reconstruction of a signal (e.g., time series data). According to CS theory [22], the signal can be reconstructed or recovered from a few samples by exploiting the sparsity characteristic of the original signal. Considering a vector signal $x \in \mathbb{R}^{n \times 1}$ having the sparsity level k (i.e., x has only $k \ll n$ non-zero elements), the compressive sensing problem is to recover the sparse vector x from the linear measurement vector $y = Ax$. $A \in \mathbf{R}^{m \times n}$ is called the measurement matrix or the coding matrix ($m \ll n$). Then, we can obtain the signal x by solving the following problem:

$$\min_{x \in \mathbb{R}^{N \times 1}} ||x|| \quad \text{subject to} \quad y = Ax \tag{2.4}$$

Problem 2.4 can be solved by using convex relaxation algorithms such as basis pursuit. For instance, CS problem can be formulated as a Lagrangian relaxation of a quadratic program [13] as:

$$\hat{x} = \arg\min_{x} ||x||_1 + \lambda ||y - Ax||_2 x \tag{2.5}$$

with $||.||_1$ and $||.||_2$ being l_1-norm and l_2-norm.

2.4 Related work

2.4.1 Traffic matrix prediction

In this section, we will summarize related studies on traffic matrix prediction problems. Network traffic prediction is an important research problem in routing optimization. The goal of traffic prediction is to estimate the future trend of traffic demands by analyzing the historical traffic information. Based on the prediction results, the network controller can calculate the proactive routing policies in advance to adapt to the dynamic traffic in near future. In this way, the controller can take appropriate actions before traffic congestion occurs and improve QoS. Therefore, by improving the prediction accuracy, we can achieve better performance in traffic engineering tasks.

Since the traffic matrix is one of the most important pieces of information used to solve the network routing problem in the backbone network, most of the existing studies focus on estimating the future traffic matrix. Originally, researchers referred to some simple statistical models such as ARIMA or Gaussian model [71]. However, such simple models cannot handle the complexity and dynamics of communication behavior in modern networks. To this end, deep learning techniques have been exploited more in predicting traffic [8, 33, 57]. In [57] and [8], the authors utilized the Long Short-Term Memory and Convolutional Neural Network for capturing the spatial-temporal feature and predicting the network traffic in data center and cellular networks, respectively. In the backbone network, Nie et al. used Restricted Boltzmann Machine to capture the dynamic features of the network. The authors then proposed two separated deep belief network model for solving the traffic estimation and prediction, independently. More recently, the results in [4] and [56] showed the superiority of the LSTM network in modeling the temporal feature and the long-range dependencies of network traffic.

Unfortunately, all the approaches proposed so far assumed that in the prediction phase, the controller can obtain all information (i.e., full traffic matrix) of the network. However, due to a large number of traffic flows, collecting all the information leads to high monitoring overhead. In this work, we consider the case that only a part of the traffic matrix is measured. The missing values in the traffic matrix are filled by using the predicted values of the previous steps. To the best of our knowledge, there is no existing work addressing the problem of traffic prediction under missing ground-truth data in the backbone network.

2.4.2 Network traffic imputation

In the past decades, various approaches have been developed to address missing values in time series. The missing values can be filled by using statistical models such as Autoregressive Moving Average (ARMA) or Autoregressive Integrated Moving Average (ARIMA) [6]. However, these models are essentially linear and process time series independently. Therefore, they fail to exploit the correlation between different variables in multivariate time series. Matrix and tensor factorization were also applied to solve the data imputation problem. Many tensor completion algorithms have been proposed based on Alternating Least Square (ALS) such as Localized Tensor Decomposition (LTC) [64], gradient-based method such as Generalized Canonical Polyadic Tensor Decomposition (GCP) [18]. Among those, LTC [64] would be seen as the most recent work on estimating missing values, tailored for network traffic data. In LTC, the traffic matrix was represented as a 3-way tensor, including days, hours, and origin-destination dimensions, to exploit the inherent relationship among higher-dimensional data. Also, LTC divides the huge 3-way traffic tensor into many sub-tensors with highly relevant data and performs Canonical Polyadic (CP) decomposition on these substructures. This approach is claimed to be more efficient than decomposing the original traffic tensor. However, all of the tensor completion algorithms do not encapsulate the temporal correlation in time series data. In particular, the imputation data is formed by summing up the outer product of component matrices. The data which is imputed by CP in each time step cannot utilize the information from the previous and the following time steps. Moreover, most of the tensor completion methods rely on a strong assumption that the tensor data has a low-rank structure.

Other approaches for estimating the missing values are deep learning-based techniques. Many imputation methods, which are based on Recurrent Neural Network (RNN) model such as Long Short-term Memory (LSTM) or Gated Recurrent Unit (GRU), achieve good results in estimating the missing values of time series and sequence data. Che et al. proposed GRU-D [10], in which the missing data were represented as the combination of the last observed values and the mean value. GRU-D laid the foundation for other methods and demonstrated its significantly high performance on health-care data with labels. Unfortunately, this method can not be directly applicable in an unsupervised manner on general datasets without the tag for each time series as our focus on network traffic data. Following GRU-D, Cao et al. introduced Bidirectional Recurrent Imputation for Time-series (BRITS) as a novel bidirectional LSTM-based model for multivariate time series recovery [7]. A temporal decay factor and a linear layer were introduced in the BRITS model, which can help learn the spatial-temporal features of data recorded in irregular intervals. More recently, Generative Adversarial Network (GAN) has been used to impute missing values. Luo et al. proposed an End-to-end GAN (E^{2}GAN) [30] for extracting feature representation of time series and reconstructing it from the low-dimensional vector. E^{2}GAN model leverages the GRU-I cell, which was used in [7], to process the incomplete time series. However, E^{2}GAN only uses a unidirectional recurrent model and does not consider the correlation between the time series. Most recently, motivated by CP decomposition, NTC model was introduced in [61], which was specifically designed for imputing network traffic. However, by taking the index of each observed data as the input, NTC suffers from a scalability issue when the number of observed data in the training set is significantly large.

Although many efforts have been devoted to data imputation, most of the deep

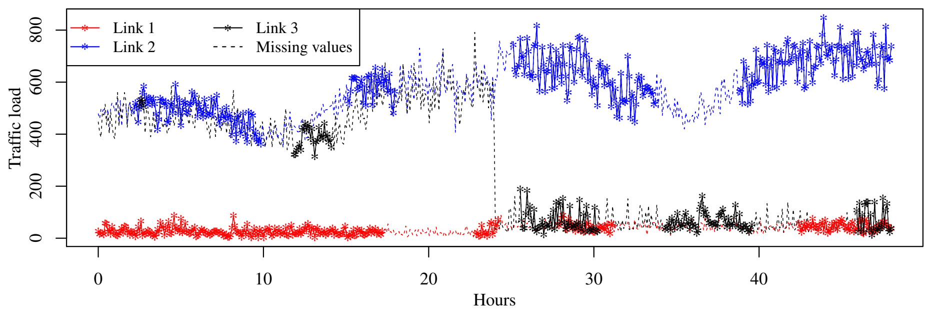


Figure 2.7: Example of network link load traffic under block missing scenarios on two different days for three links in the Abilene network. The dot lines represent the missing blocks, while the stars indicate the observed data.

learning methods proposed so far have not focused on the network traffic data. Therefore, they suffer from the following critical problems. First, most of the methods mainly focused on the temporal relationship. Some approaches (e.g., BRITS [7]) have considered leveraging the spatial features, but they only cope with the static spatial correlations. Meanwhile, the correlations between the time series in the network traffic data vary significantly over time due to network behavior dynamics. Fig.2.7 visualizes the traffic load on three different links in the Abilene network to show the variation of the network behavior. Due to the dynamic of the routing scheme, the correlation between the links' traffic is inconsistent. Specifically, the traffic on link 3 shows a high correlation to that of link 2 on the first day, whereas, on the next day, it appears to be closely similar to link 1. Second, the existing models have not been evaluated by the network traffic dataset (e.g., the Abilene dataset). As the network traffic possesses unique characteristics, as mentioned above, the evaluation results for other data types are not likely to fit with network traffic data.

2.4.3 Traffic engineering with segment routing

Here, we present an overview of the current SR-based TE solutions. Due to the routing flexibility, SR has been well investigated in both theoretical analysis and practical approaches. In [5], the TE problem with segment routing is formulated as a linear programming problem in which only two segments were considered (by choosing one intermediate node between the source and destination nodes). Later on, several works focused on utilizing more than two segments. For example, the authors in [37] proposed an optimization model which uses three segments and combines both node and edge segments. In [21], the authors attained a further improvement on the TE problem with SR by fully exploiting the SR (n-segment routing with both node and edge segments) and proposed the first Column Generation-based approach. In [28] and [38], the authors used the SR to enhance the routing management in Software Defined Networks. They proposed efficient routing algorithms that can solve the scalability issues and improve traffic distribution.

For the practical approach, in [14], the authors considered both the unexpected traffic fluctuation and link failure problems. They proposed a local search-based algorithm to solve the targeted problems under sub-second constraint. Recently, the authors in [45, 46] extended the work in [5] by proposing an optimization model to minimize the number of deployed SR policies.

Although the TE problem with the Segment Routing issue has been well studied, most of the solutions proposed so far only addressed the local optimization, where they considered the problem in only a single snapshot. Specifically, the primary approach is to formulate the problem under a mixed-integer linear programming model, and then propose heuristic algorithms using various techniques such as local search (**srls**) [14], column generation (**cg4sr**) [21]. This approach usually used the traffic matrix (i.e., traffic demands) of the corresponding snapshot as the input. However, due to the network behavior's dynamic, the traffic matrix often varies over the snapshots; thus, leading to the changes in the routing policy obtained. In some rare work [5], the authors proposed a Traffic Matrix Oblivious Segment Routing which does not require the traffic matrix to be given prior. The proposed routing policy was designed to work well for a wide range of traffic demands. Although this approach can alleviate the overhead caused by routing path change, it may not guarantee the link utilization constraint, especially with the network's dynamic behavior. Besides, it was pointed out in [14] that the Traffic Matrix Oblivious Segment Routing has been showed to only be practical for offline traffic engineering on relatively-small networks.

3 Traffic matrix prediction with partial information

Accurate prediction of the future network traffic plays an important role in various network problems (e.g. traffic engineering, capacity planning, quality of service provisioning, etc.). Although the prediction accuracy largely depends on the quality of historical data, obtaining high quality data by measuring all the network traffic is impossible or impractical due to the monitoring resources constraints as well as the dynamics of temporal/spatial fluctuations of the traffic.

In this chapter, we propose novel DNN-based model for traffic prediction under the lacking of network traffic data. We first introduce the problem statement in Section 3.1. Then, we present the challenges in Section 3.2. In Section 3.3 and 3.4, we give the details of our proposed prediction model and the strategy for selecting monitored flows. We show the performance evaluation of our methods in Section 3.5 and summarize the contributions in Section 3.7.

3.1 Problem statement

First, we will describe the network model, as well as terms and notations. The network is represented by a directed graph $G = (V, E)$, where V is the set of nodes ($|V| = N$), and E is the set the network's links. Each link $e \in E$ has capacity $c(e)$. Let $X_t \in \mathbb{R}^{N \times N}$ be the traffic matrix at time-step t and $x^t_{sd} \in X_t$ denote the traffic flow from node s to d (flow sd for short, $s, d \in V$) at the time-step t. The term "flow sd" indicates the aggregated traffic that enters the network at node s and exits at node d. The traffic matrix prediction problem is to estimate the traffic matrices of next T time-steps (denoted by $\widetilde{X}_{t+1}, ..., \widetilde{X}_{t+T}$), given the previous H measurements ($T, H \geq 1$):

$$\widetilde{X}_{t+1}, ..., \widetilde{X}_{t+T} = \underset{X_{t+1},...,X_{t+T}}{\operatorname{argmax}} \; p(X_{t+1}, ..., X_{t+T} | X_{t-H+1}, ..., X_t) \tag{3.1}$$

Due to the high monitoring overhead, we consider the case of backbone networks where we cannot obtain all the H previous traffic matrices by directly monitoring all the flows. The monitoring ratio ω of the network is defined in Equation 3.2. k is the number of monitored flows; $N \times N$ is the total number of flows in the network.

$$\omega = \frac{k}{N \times N} \tag{3.2}$$

To achieve low monitoring overhead, we choose to reduce the monitoring ratio by keeping the value of ω less than 1 which means, at every time-step, only a part of total flows is observed and then the missing data (the flows without being monitored) is filled by the predicted value. We call this type of network monitoring as *partial monitoring*. To use partial monitoring as the input data for future traffic prediction, the missing data (the flows without being monitored) is filled by the predicted value. Therefore, in the traffic matrix prediction with partial information, the inputs contain both observed data and predicted data, which is obtained from the last prediction step. Accordingly, the traffic matrix prediction with partial information problem is formulated as follows:

Input

$$x_i^{s,d} = \begin{cases} o_i^{s,d} & \text{if } m_i^{s,d} = 1 \\ \hat{x}_i^{s,d} & \text{otherwise} \end{cases} \tag{3.3}$$
$$\forall s, d \in V; i = t - H + 1, ..., t$$

Output

$$\hat{X}_{t+1},...,\hat{X}_{t+T} = \underset{X_{t+1},...,X_{t+T}}{\operatorname{argmax}}\ p(X_{t+1},...,X_{t+T}|X_{t-H+1},...,X_t) \tag{3.4}$$

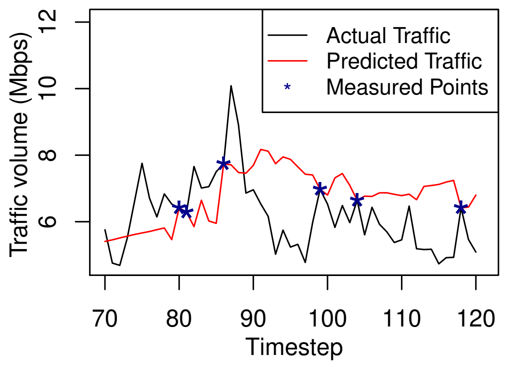
where $o_i^{s,d}$ and $\hat{x}_i^{s,d}$ denote the observed and predicted traffic volume of flow (s,d) at time-step i, respectively. The binary variable $m_i^{s,d}$ depends on the monitoring strategies, where $m_i^{s,d} = 1$ indicates that flow (s,d) is monitored at time-step i; otherwise, the value of traffic volume is filled up by the prediction result.

In the so-called partial monitoring, we can randomly determine the flows that are measured at every time-step or follow a pre-designed strategy. Thank the advances of new network architectures and technologies such as Software-Defined Networking (SDN) [31], In-band Network Telemetry (INT) [16],[19], the network operator now can to measure, retrieve the flow information, and collecting the traffic statistics. We will discuss the effect of the monitoring strategy on the prediction accuracy in Section 3.4.

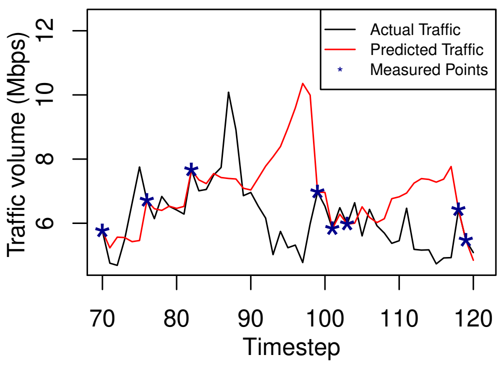
3.2 The low accuracy of traffic matrix prediction with partial information

We have performed experiments to figure out the impact of imprecise input data on the results of one-step-ahead prediction (i.e., $T = 1$). In this experiment, we assume that the network controller can only measure a subset of traffic flows. The monitoring ratio is set from 10% to 40%. We train a simple LSTM network to estimate the traffic matrix of the next time-step. The input for the LSTM network is the historical traffic matrices. However, since the controller cannot measures all traffic flows, we fill the missing values by the predicted values of the previous prediction step. The low monitoring ratio leads to the higher number of the imprecise values in the input sequence, hence the higher prediction error. As the error in the input data results in the error in the prediction's output, we call this problem as the accumulative error.

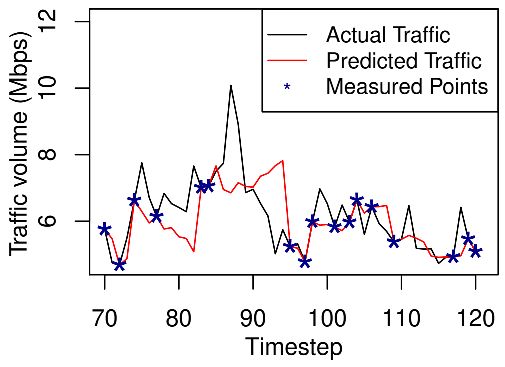
Fig.3.1 shows the one-step-ahead prediction results with various settings of the monitoring ratio. As showed, the accuracy tends to decrease with the increasing of the time-steps between two consecutive measured points (i.e., Fig.3.1(a), Fig.3.1(b)). While in Fig.3.1(d), thanks to the higher percentage of ground-truth data in the input, we



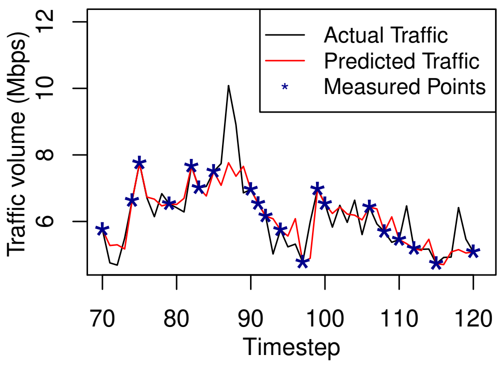
(a) 10% monitoring ratio.



(b) 20% monitoring ratio.



(c) 30% monitoring ratio.



(d) 40% monitoring ratio.

Figure 3.1: The effect of monitoring ratio on prediction accuracy (One-step-ahead prediction using the LSTM network).

can well capture the trend of the flow and achieve high accuracy in forecasting the future traffic. Therefore, to alleviate the accumulative error while remaining the low monitoring overhead (which is proportional to the portion of the ground-truth data), our idea is to perform preprocessing on the imprecise data before feeding it into the prediction model. Specifically, we propose a novel deep learning model based on the bidirectional recurrent neural network (the details will be shown in Section 3.3.3).

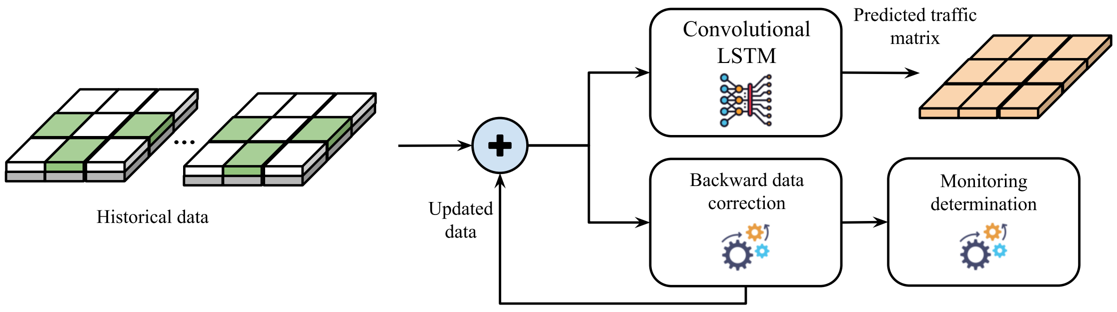


Figure 3.2: The overall system of traffic matrix prediction with partial information.

3.3 ConvLSTM-based traffic matrix prediction

3.3.1 Overview of the proposed approach

Figure 3.2 gives the overview of our framework for monitoring the network partially and predicting future traffic matrix based on partial information. The proposed approach includes three main modules: traffic matrix prediction, data correction, and monitored flow determination. The network monitoring function, which measures and/or collects the traffic data from the network (based on the monitoring strategy), is out of the scope of this work.

- **The traffic matrix prediction** The traffic matrix prediction module takes H traffic matrices from time-step $t - H + 1$ to t as input and gives the outputs the next T time-steps (i.e., $t + 1, ..., t + T$). The details of the traffic matrix prediction module are described in Section 3.3.2.

- **Backward data correction** To improve the prediction accuracy, the backward data correction module leverages the information in the output of the prediction module and the current input to correct the imprecise data in the input. Then the updated data is continuously used in future prediction (see Section 3.3.3).

- **Monitored flow determination** The monitored flow determination decides appropriate flows to be monitored in the next time-step. Our determination strategy is given by following a heuristic approach, which aims to increase the traffic forecasting accuracy while remaining a low monitoring overhead. Specifically, after every time-step, we calculate a weight for each traffic flow by using the prediction accuracy and the monitoring statistic. The weight is used to decide the flows which will be monitored. The details of the determination strategy are presented in Section 3.4.

3.3.2 Future traffic matrix prediction

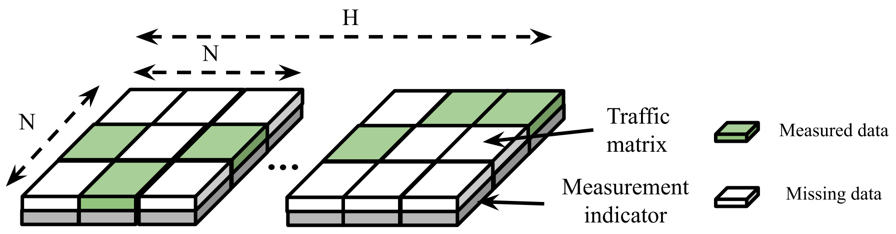


Figure 3.3: The sequence of H input matrices of ConvLSTM network.

Although having the advantages in terms of usage memory and computation, the LSTM model forecasts traffic flows independently, and thus, it can extract only the temporal feature of time series data. Therefore, by using the LSTM network, we may not exploit the spatial relation between the traffic flows which have been figured out in [63]. To this end, we also consider using the Convolutional LSTM network (ConvLSTM) which can extract both temporal and spatial features of the input sequences.

Convolutional LSTM network [65] (ConvLSTM) is a combination of convolutional operation and LSTM network. While the LSTM network only takes a 1D array as the input at each processing step, the ConvLSTM can extract spatio-temporal features from a sequence of 2D or 3D tensors.

To apply the ConvLSTM network in our problem, we consider the traffic matrix as a 2D image with the dimension of $N \times N$, and then we transform them into 3D tensors with the dimension of $(N, N, 2)$ by combining the measurement matrix M and the traffic matrix. Accordingly, the input of our ConvLSTM network has dimension $(H, N, N, 2)$ which is a sequence of H 3D tensors (Figure 3.3).

Figure 3.4 illustrates a Convolutional LSTM network for traffic matrix prediction which contains only one convolutional layer. To construct a deep and complex model, we can stack other layers on top of the previous ones.

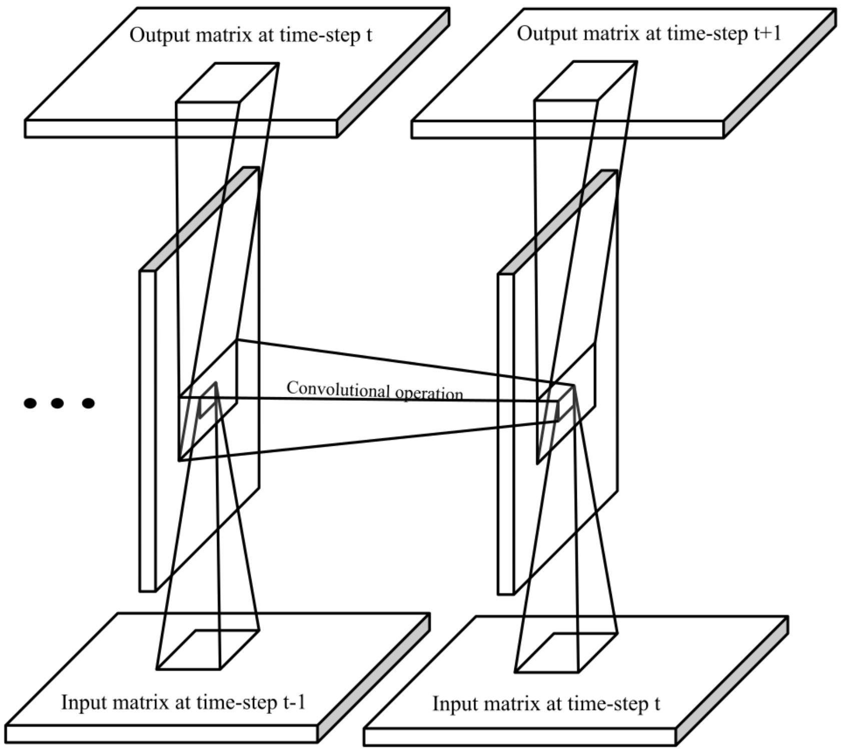

Output matrix at time-step t
Output matrix at time-step t+1
Convolutional operation
Input matrix at time-step t-1
Input matrix at time-step t

Figure 3.4: The structure of ConvLSTM network for traffic matrix prediction.

3.3.3 Backward data correction

As described in Section 3.2, the imprecise data in the input has a huge impact on the prediction accuracy. In this section, we will describe our method to overcome this problem by correcting the imprecise data. Following the experiment in Section 3.2 (Fig.3.1), we observe that the prediction results become more accurate if the input sequence contains more precise data. Besides that, the accuracy of the predicted traffic at a time-step whose previous data is ground-truth data is better than the others. Based on the above observations, we propose an algorithm which uses the outputs from each processing step of the ConvLSTM network and the measurement matrices to correct the imprecise data. To be more specific, considering an example where we predict the future traffic matrix at time-step $t + 1$ by taking the sequence $\{X_{t-H+1}, ..., X_t\}$ as the input. As using the many-to-many model, after the prediction, we also obtain the outputs $\{\hat{X}_{t-H+2}, ..., \hat{X}_{t+1}\}$ whose each element corresponds to one processing step of the ConvLSTM network. Suppose that $x_i^{s,d} \in X_i$ $(t - H + 2 \leq i \leq t)$ is an imprecise

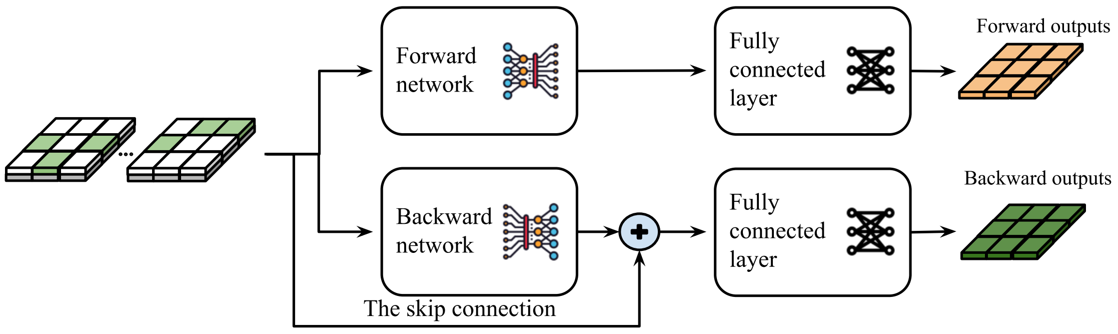


Figure 3.5: The forward and backward networks with skip connection.

data and $\hat{x}_i^{s,d} \in \hat{X}_i$ is its corresponding output in the ConvLSTM network. Because of the forward direction in the processing step of the ConvLSTM network, the output $\hat{x}_i^{s,d}$ is generated by taking only the input from time-step $t - H + 1$ to $t - 1$. If this input sequence contains almost ground-truth data, we can imply that $\hat{x}_i^{s,d}$ may close to the ground-true value than the current $x_i^{s,d}$ ($|\hat{x}_i^{s,d} - o_i^{s,d}| < |x_i^{s,d} - o_i^{s,d}|$). Thus, we may replace $x_i^{s,d}$ by $\hat{x}_i^{s,d}$.

However, we face the problem when the input sequence from time-step $t - H + 1$ to $i - 1$ contains only a few ground-truth data. In this case, $\hat{x}_i^{s,d}$ may not close to $o_i^{s,d}$ compared to $x_i^{s,d}$. To this end, our idea is to leverage the data of the next time-steps (from $i + 1$ to t) which may include more precise data. To implement the above idea, besides the current ConvLSTM network we construct an extra network in which the sequence of input data is fed in the reverse order. To distinguish the two ConvLSTM networks, the first network is called *forward network* and the additional network is called *backward network*. Our approach is motivated by the BiLSTM which was introduced in [47]. Thanks to adding a backward network, BiLSTM can be trained by all available input information in both the past and the future of a specific time frame. This technique is well known for predicting the missing word in the sentence in the Natural Language Processing problem. The BiLSTM can acquire the context before and after the missing position to improve accuracy.

Different from the standard BiLSTM, in each processing step, instead of aggregating the outputs of the forward and backward networks, we keep them separately (as showed in Fig.3.5), and use the outputs from the backward network (denoted as $\tilde{x}_i^{s,d}$) to update the incorrect data in the input matrices. Therefore, instead of simply replacing the $x_i^{s,d}$ by $\hat{x}_i^{s,d}$, now we can update it by using the outputs of backward network (i.e., $\tilde{x}_i^{s,d}$) which may be predicted using more precise data.

3.3.4 The skip connection

When using the backward outputs to update the imprecise value, the key is the accuracy of the values generated by the backward network. If the outputs of the backward network have low accuracy than the current input, the updating may be fail. While in the forward network, only the output for the future time-step (i.e., $t+1$) is matter, in the backward network, all the outputs are important since they participate in the data correction.

To further increase the accuracy of the backward network, we introduce a skip connection from the input to the Fully Connected Network as showed in Figure 3.5. The input , which includes the traffic matrix and the measurement matrix, is concatenated with the result from the backward ConvLSTM layer before go to the Fully Connected Network. By doing this, we can slightly improve the performance of the backward network. We have conduct experiments to evaluate the effect of the skip connection in Section 3.6.3.

3.3.5 The backward data correction algorithm

The updated value of the imprecise data can be calculated by taking the average from the current value and the backward network output. However, it may decline the accuracy when the backward network output is generated using imprecise data. To avoid that, instead of taking the average, we use the weighted average. In order to determine the contribution of the current value $x_i^{s,d}$ and backward network output $\tilde{x}_i^{s,d}$ in updating the imprecise data, we define parameters $\alpha_i^{s,d}$ which is the confidence factors of $\tilde{x}_i^{s,d}$ $(0 \leq \alpha_i^{s,d} \leq 1)$. The details of the imprecise data correction algorithm is described in Algorithm 1 and illustrated by Fig. 3.6. Note that since the outputs of the backward network are from $t-H$ to $t-1$ while the input is from time-step $t-H+1$ to t, we can only apply Algorithm 1 for correcting the imprecise data from time-step $t-H+1$ to $t-1$. In what follows, we describe the main idea behind Algorithm 1.

The less the ground-truth data contained in the inputs, the less precise the predicted values, thus $\tilde{x}_i^{s,d}$ should not be updated if the historical data is highly missed. In

Algorithm 1: Backward data correction

```
input  : $X_i$: the previous traffic matrix at time-step $i$
         $\tilde{X}_i$: the outputs of backward network
         $M_i$: the measurement matrix of $X_i$
         $(i = t - H + 2, ..., t - 1)$
output : The updated traffic matrices
1  for $s, d \in \mathcal{N}$ do
2      for $i = t - H + 1$ to $t - 1$ do
3          if $i{+}r > t$ then
4              $\alpha_i^{s,d} \leftarrow 0.0$;
5          end
6          else
7              $\mu_i^{s,d} \leftarrow \frac{\sum_{j=1}^{r} m_{i+j}^{s,d}}{r}$;
8              $\rho_i^{s,d} \leftarrow \frac{1}{\ln(r)+1} \times \sum_{j=1}^{r} \frac{m_{i+j}^{s,d}}{j}$;
9              $\alpha_i^{s,d} \leftarrow \mu_i^{s,d} \times \rho_i^{s,d}$;
10         end
11         $x_i^{s,d} \leftarrow (1.0 - \alpha_i^{s,d}) \times x_i^{s,d} + \alpha_i^{s,d} \times \tilde{x}_i^{s,d}$;
12     end
13 end
14 return $X_i$ $(i = t - H + 2, ..., t - 1)$
```

addition, based on the experiment in Section 3.2, the predicted values are more close to the ground-truth value if the previous time-step's data is the precise data. Therefore, we consider the position of the precise data in the input which is used to generate $\tilde{x}_i^{s,d}$ (i.e., $x_{i+1}^{s,d}, ..., x_t^{s,d}$). However, some output of the backward network is generated using too few data, for example the $\tilde{x}_{i-2}^{s,d}$ is generated using the information from only two data (i.e., $x_{i-1}^{s,d}$ and $x_i^{s,d}$). Therefore, when calculate the confident factor $\alpha_i^{s,d}$, we only consider the output that is generated by using r data ($1 < r < H$).

Accordingly, in Algorithm 1, we first calculate the monitoring ratio (i.e., $\mu_i^{s,d}$) in the input $x_{i+1}^{s,d}, ..., x_t^{s,d}$ at line 7. Then, we use the equation in line 8 to assess the position of the precise data in the input (i.e., $\rho_i^{s,d}$). After that, the confident factor $\alpha_i^{s,d}$ is calculated as following:

$$\alpha_i^{s,d} = \frac{1}{\ln(r) + 1} \times \frac{\sum_{j=1}^{r} m_{i+i}^{s,d}}{r} \times \sum_{k=1}^{r} \frac{m_{i+k}^{s,d}}{k} \tag{3.5}$$

We have $\alpha_i^{s,d} \leq \frac{1}{\ln r+1} \times H_r < 1$ where $H_r = 1 + \frac{1}{2} + ... + \frac{1}{r}$ is the sum of Harmonic series. By using equation (3.5), we guarantee that $\alpha_i^{s,d}$ is always less than 1.0 even in the case all the inputs are precise.

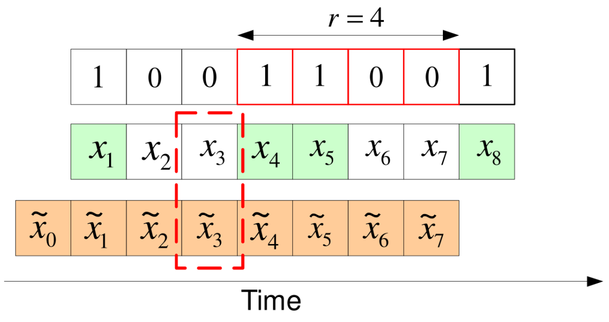


Figure 3.6: Example of data correction.

In Figure 3.6, we show an example of the data correction algorithm. In this example, a traffic flow is predicted using 8 time-steps of input and we consider $r = 4$. Suppose that we want to update the value of imprecise data x_3 (the (s, d) is removed for simplicity). Then, the confident factors are calculated as follows:

$$\mu_3 = \frac{\sum_{j=1}^{r} m_{i+j}}{r} = \frac{1}{2}$$
$$\rho_3 = \frac{1}{\ln r+1} \times \frac{\sum_{j=1}^{r} m_{i+j}}{r} \times \sum_{k=1}^{r} \frac{m_{i+k}}{k} = \frac{1}{\ln 4+1} \times \frac{3}{2}$$
$$\alpha_3 = \mu_3 \times \rho_3 \approx 0.314$$

Accordingly, the updated value are calculated as $x_3 \leftarrow (1 - \alpha_3) \times x_3 + \alpha_3 \times \tilde{x}_3$.

3.4 Monitored flow determination

The monitored flow determination is the mechanism used for selecting the subset of flows that is observed after every time-step. One of the common approaches in choosing the set of monitored flows is to satisfy the fairness among all the flows. Specifically, the gaps between every two consecutive monitored time-steps of every flow are kept to be approximately the same. However, this method may not be effective since the network flows are dynamic and have various temporal fluctuation patterns. In this section, we design a heuristic monitoring strategy called weighted monitoring which respects two factors: the recent monitoring statistic and the prediction accuracy. Our main idea is that at the current time-step t, we calculate a weight $w_t^{s,d}$ for each flow (s, d) and choose at most k flows which have the lowest weights to be monitored at the next time-step (i.e., time-step $t + 1$). The maximum number of flows that can be monitored depends on the monitoring ratio (i.e., $k = \lfloor \omega_{t+1} \times N \times N \rfloor$).

In the following, before going to the details of the weight's formula, we will first describe our theoretical basis. To ease the presentation, we call the periods between the consecutively measured time-steps of a flow non-monitored periods of that flow. Following the experiment results shown in Section 3.2, we note that the longer the non-monitored periods, the larger the difference between the predicted results and the actual traffic. Thus, to reduce the prediction error, we should decrease the non-monitored periods of all flows. More specifically, the flows that have not been monitored for a long period should be chosen to be monitored at the next time-step. To this end, for each flow (s, d), we define a term named *consecutive missing* (denoted as $c_t^{s,d}$) which is the number of the time-steps from when (s, d) was last monitored till the current time-step t. The weight should be designed so that it will decline when the consecutive missing gets high.

Since we do not have the observed data at the time the prediction is made, the prediction accuracy cannot be calculated. Therefore, to evaluate the prediction accuracy, we define two terms named *forward loss* and *backward loss*. The forward and backward losses of a flow (s, d) (denoted as $l_t^{s,d,f}$ and $l_t^{s,d,b}$, respectively) are defined to assess the performance of the forward and backward ConvLSTM networks after predicting the traffic at current time-step t. Specifically, $l_t^{s,d,f}$ and $l_t^{s,d,b}$ are defined as the root squared errors between the outputs and the ground-truth elements in the input (Equations (3.6) and (3.7)). Note that, if the input contains no ground-truth data, then the losses are assigned to ξ which is a large positive number.

$$l_t^{s,d,f} \leftarrow \frac{1}{\sum_{i=t-H+2}^{t} m_i} \sum_{i=t-H+2}^{t} m_i^{s,d} \times (\hat{x}_i^{s,d} - x_i^{s,d})^2 \tag{3.6}$$

$$l_t^{s,d,b} \leftarrow \frac{1}{\sum_{i=t-H+1}^{t-1} m_i} \sum_{i=t-H+1}^{t-1} m_i^{s,d} \times (\tilde{x}_i^{s,d} - x_i^{s,d})^2 \tag{3.7}$$

Consequently, the weight is calculated based on the flows' consecutive missing, backward loss, and forward loss.

$$w_t^{s,d} = \frac{1}{\lambda_1 \times l_t^{s,d,f} + \lambda_2 \times l_t^{s,d,b} + \lambda_3 \times c_t^{s,d}} \tag{3.8}$$

where λ_1, λ_2 and λ_3 are hyper-parameters which are chosen by experiments.

At the end of the time-step t, the weights of all flows are calculated, and the first k flows with the lowest weights are chosen to be measured at the next time-step.

3.5 Performance evaluation

3.5.1 Evaluation metrics

We evaluate the performance of our proposed model FWBW-LSTM and compare it with two different models: statistical model ARIMA and LSTM network. We use the error ratio (ER) as the performance metric to evaluate the prediction accuracy. Besides that, we also compare the training and the prediction time of all approaches.

- **Error ratio** (the lower is better) is a metric for measuring the accuracy of the predicted value. Since the traffic matrices include both observed and predicted data, we only consider calculate the error of the predicted data (data in which $m_t^{s,d} = 0$). ER can be calculated as follows.

$$ER = \frac{\sqrt{\sum_{s,d \in \mathcal{N}} \sum_{i=1}^{H} (1 - m_i^{s,d}) \times (x_i^{s,d} - o_i^{s,d})^2}}{\sqrt{\sum_{s,d \in \mathcal{N}} \sum_{i=1}^{H} (1 - m_i^{s,d}) \times (o_i^{s,d})^2}} \tag{3.9}$$

In equations (3.9), H is the total number of time-steps in the test set, and $o_i^{s,d}$ is the ground-truth traffic volume of flow (s, d) at time-step i.

3.5.2 Dataset

We evaluated the performance of our proposed approach by conducting extensive experiments on the Abilene dataset (available at [70]). In the experiments, we separated the dataset into 60% for training, 20%, and 20% for testing and validating, respectively.

LSTM network configurations			
No. LSTM layers	1	Recurrent dropout	0.5
No. hidden units	128	No. FC layers	3
ConvLSTM network configurations			
No. Convolutional layers	2	No. filters	4/layer
Stride	(1,1); (1,1)	Filters' size	(3,3); (5,5)
Convolutional dropout	0.5	Recurrent dropout	0.5
Training configurations			
Batch size	512	Loss function	mse
Optimizer	adam	No. trained epoches	50

Table 3.1: The configurations of LSTM and ConvLSTM networks.

- **Abilene dataset** is the real trace data from the backbone network located in North America which contains 12 nodes ($n = 12$). Abilene dataset, which includes averages over 5 minutes interval of 144 aggregated flows from March 1 to September 11, 2004, has been widely used for performance evaluation in many traffic matrix prediction studies [62], [63].

3.5.3 Model setting

The experiments have been conducted on a computer that has Intel i7-6900K CPU @ 3.20GHz, 64 GB memory, and two NVIDIA GeForce GTX 1080Ti. The detailed configurations of the LSTM and Convolutional LSTM networks are listed in Table 3.1. For the ARIMA model, we use the "pyramid-arima" library. The library allows us to quickly perform this grid search and even creates a model object that you can fit the training data.

3.6 Experimental results

We conducted two types of experiments. First, we evaluated the performance of all the three algorithms in one-step-ahead traffic matrix prediction ($T = 1$). In the second experiment, we conducted a multi-step-ahead traffic matrix forecasting by predicting the traffic matrices of 15 minutes ahead of the current time-step ($T = 3$). In order to

	Input size (H)	Prediction steps (T)	Weighted monitoring parameters (λ_1, λ_2, λ_3)	No. run time
One-step-ahead prediction	30	1	(2.6, 1.0, 1.0)	50
Multi-step-ahead prediction	30	3, 6, 9	(2.6, 1.0, 1.0)	50

Table 3.2: Experiment configurations.

perform the multi-step traffic matrices prediction, we apply the Iterated Multi-Step estimation (IMS) approach [48]. Specifically, we first use the many-to-many model to predict the traffic matrix of the next time-step and then, iteratively feed the generated data into the model to get the multi-step future traffic matrices.

In each experiment, we conducted different scenarios in which the monitoring ratio ω is varied from 10% to 90% (i.e., the maximum number of monitored flows per time-step is $k = \omega \times N \times N$). In the experiment, we also applied the different monitoring strategies for each model: random, fairness and weighted monitoring (note that the weighted monitoring strategy is only used for FWBW-LSTM model). Therefore, the results will be showed as following: {model name}-{monitoring strategy} (e.g., LSTM-FAIRNESS stands for the result of LSTM model using fairness monitoring).

Table 3.2 shows the configuration of each experiments. For each experiment using random monitoring strategy, we run the experiment 50 times and calculate the average results.

3.6.1 Comparison of LSTM and ConvLSTM network-based traffic matrix prediction

In this part, we compare the performance of LSTM and Convolutional LSTM on one-step-ahead traffic prediction in term of Error Ratio. In the experiments, we apply the random monitoring for both approach and vary the monitoring ratio from 10% to 90%.

The figure 3.7 indicates that in all the cases of monitoring ratio, LSTM has outperformed the ConvLSTM. LSTM has 25% less Error Ratio than the figure of ConvLSTM in average. There are several reasons for the poor performance of

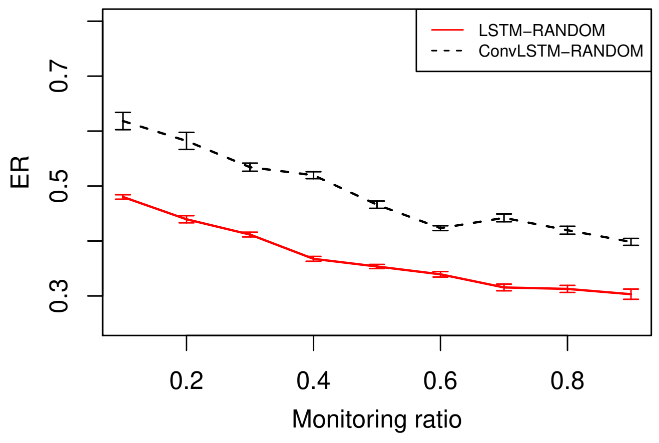


Figure 3.7: The comparison between LSTM and ConvLSTM-based approach.

Convolutional LSTM network-based approach. First, by adding the convolutional operation, the Convolutional LSTM network has become more complicated and difficult for training and hyper-parameter tuning (e.g., selecting the number of filters, filters' size, etc.). Second, the traffic matrix is formed arbitrarily without considering the geometrical location of the node. Therefore, the strong correlated flows may not be located close to each other in which the spatial features can be extracted.

Besides that, the traffic flows in the Abilene dataset may not have strong relations. For example, we have calculated the pairwise correlation between 25 flows which are in the 5×5 square matrix at the top left of the $N \times N$ Abilene traffic matrix (we used the data in the first week of the Abilene dataset). Figure 3.8 shows the pairwise correlation of 25 flows as a heat map. As we can see, most of the flows do not have low correlations to others. Therefore, for other results presented in this work, we only consider the LSTM based-approach.

3.6.2 Evaluating the data correction module

To evaluate the performance of the data correction module, we conducted the one-step-ahead prediction and compared the results of the LSTM model and our approach both using random monitoring.

Figure 3.9 shows the comparison between the LSTM model and our proposed model. Overall, with the data correction module, we achieve a lower error ratio in

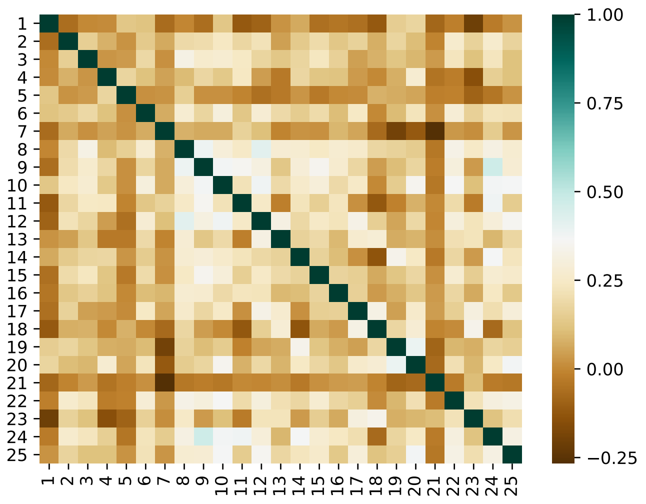


Figure 3.8: The pairwise correlation of 25 traffic flows in the 5×5 matrix.

the prediction and a higher R2 score than the standard LSTM model. The figure 3.9 indicates that with the backward data correction algorithm, we can reduce the impact of the imprecise data. The data correction module performs well in medium and high monitoring ratios. Especially, our model achieves 15.66% less error than the LSTM at 60% monitoring ratio.

The idea of the data correction module is based on the backward network and the precise data in the input to correct the imprecise one. Therefore when there are few precise data, our model only makes a slight improvement. As shown in the figure, the gap in the Error Ratio between LSTM and our approach is small in the low monitoring ratio case (ω equals 10%).

3.6.3 Evaluating of the skip connection module

In Section 3.3.4, we introduced the skip connection that allows the input directly concatenates with the output of the LSTM layer. In order to evaluate the effect of the skip connection, we conducted experiments in one-step-ahead prediction and compared the Error Ratio between the FWBW-LSTM model with and without the skip connection. In this experiment, we used the weighted monitoring strategy for both

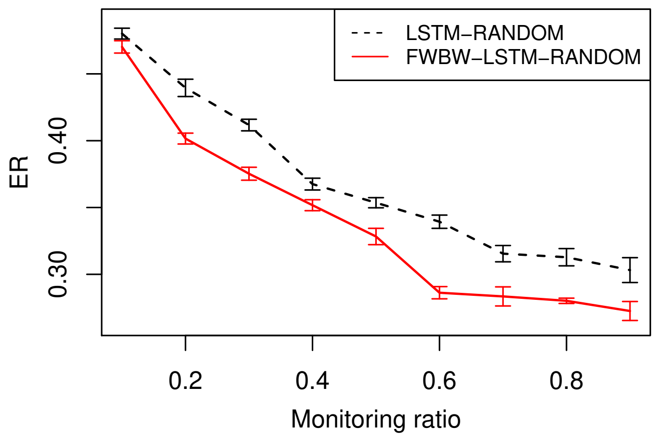


Figure 3.9: The effect of data correction module.

models and varied the monitoring ratio from 10% to 90%.

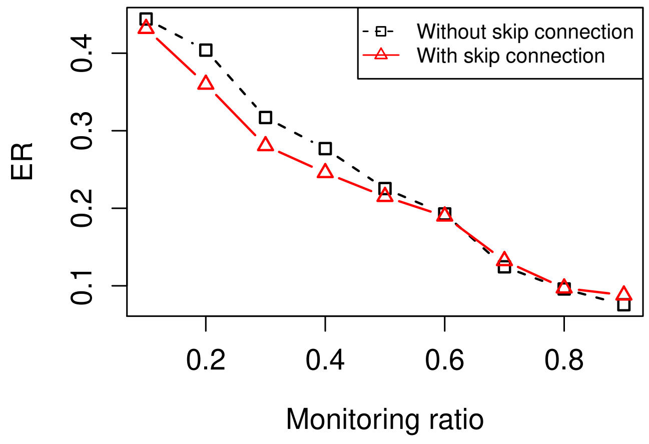


Figure 3.10: Comparison FWBW-LSTM model with and without skip connection.

In overall, the skip connection shows the advantage in the cases of highly lacking ground-truth data. The skip connection outperforms in low monitoring ratio cases (by reducing 10.9%, 11.5%, and 11,2% error in the cases ω = 20%, 30% and 40%, respectively) and achieves as same as the results of FWBW-LSTM without skip connection in high monitoring ratio.

3.6.4 Evaluating the monitored flow determination module

In Section 3.2, we have mentioned the problem in which it is necessary to have a monitoring strategy. The monitored flow determination module determines the subset of flows that need to be monitored at every time-step to obtain the ground-truth data. We have designed a simple heuristic method called weighted monitoring strategy in Section 3.4. In this section, we present the performance comparison between three monitoring strategies: random, fairness, and weighted monitoring. The experiments were conducted using the FWBW-LSTM model with the increase in monitoring ratio from 10% to 90%.

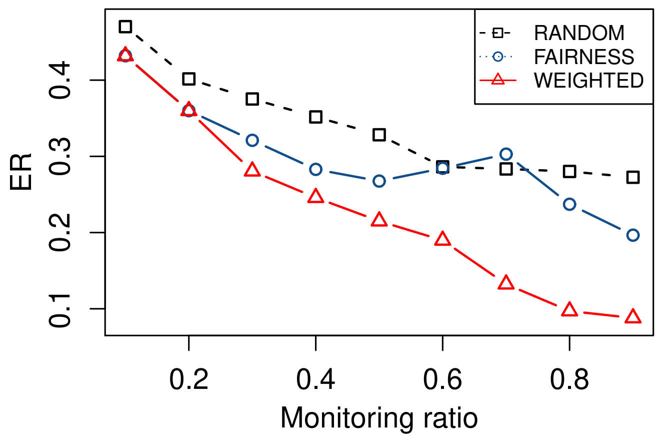


Figure 3.11: The effect of monitoring strategies on traffic matrix prediction.

Figure 3.11 shows the effectiveness of the weighted monitoring in significantly reducing the Error Ratio. Weighted monitoring achieves 68% and 60% less error than the random and fairness monitoring in the best case. In the worst-case (i.e., ω equals 10% and 20%), weighted monitoring has the same performance as fairness monitoring.

In addition, although the results of the fairness strategy are better than the random strategy in general, its performance decreases in the cases 60% and 70%. Especially, in the cases of 70% monitoring ratio, the Error Ratio of fairness monitoring is witnessed higher than that of random monitoring. The answer to this phenomenon is due to the variation in the temporal fluctuation of the flows, while the fairness strategy only ensures the same monitoring ratio for every flow in the network.

ω	10%	20%	30%	40%	50%	60%	70%	80%	90%
ER	3.47e9	3348.589	1.322	1.209	0.553	0.497	0.351	0.349	0.331

Table 3.3: The results of ARIMA model in one-step-ahead prediction.

3.6.5 Evaluating the performance of one-step-ahead prediction

In this section, we present the performance comparison between our proposed model (i.e., FWBW-LSTM with weighted monitoring strategy) and other benchmarks (i.e., ARIMA and LSTM). We conducted the one-step-ahead prediction ($T = 1$) and varied the monitoring ratio from 10% to 90%. Note that since the values regarding some ARIMA's results are extremely large and lie outside the boundaries of the figures, they are also presented in Table 3.3.

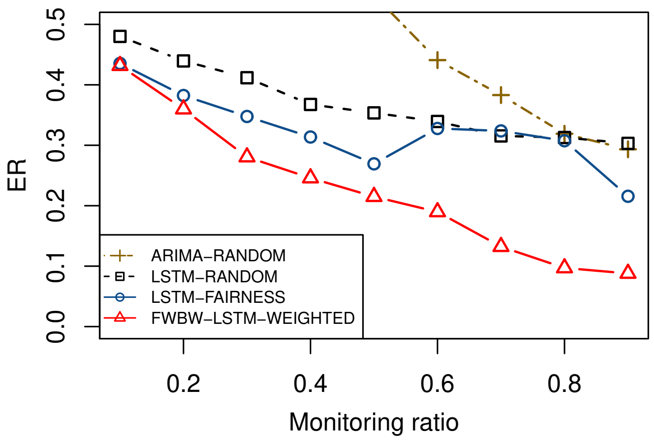


Figure 3.12: Performance comparison in one-step-ahead prediction.

It can be seen that our proposal achieves the best performance regarding all the metrics. LSTM shows the second-best performance and ARIMA is the worst. In general, the more information we have (i.e., the higher monitoring ratio), the better prediction accuracy we get. However, considering the increase of 80% in the monitoring ratio from 10% to 90%, while our approach performs 79.6% improvement in the Error Ratio, LSTM-FAIRNESS and LSTM-RANDOM models only gain 50.4% and 36.8%, respectively. Additionally, although the results of the ARMIA model are witnessed a significant improvement in reducing the error, the ARIMA model has performed poorly when lacking information ($\omega \leq 50\%$).

ω	10%	20%	30%	40%	50%	60%	70%	80%	90%
ER	1.8e9	94.166	4.780	1.095	0.861	0.613	0.748	0.546	0.710

Table 3.4: The results of ARIMA model in multi-step-ahead prediction.

Comparing LSTM and our proposal, we can see that our approach achieves high accuracy in most of the experiment scenarios. Specifically, in the best case (i.e., $\omega = 90\%$), the Error Ratio of our proposed approach is 71% and 60% less than that of LSTM-RANDOM and LSTM-FAIRNESS, respectively. Moreover, it can be seen that our proposal in case $\omega = 30\%$, achieves better performance (regarding all metrics) as LSTM-RANDOM in case $\omega = 90\%$. It can be implied that with our approach we may reduce 60% of the network monitoring overhead while still achieving the same prediction accuracy as LSTM-RANDOM does.

The ratio of ground-truth input has a massive impact on the prediction accuracy of ARIMA. As showed in Table 3.3, when $\omega < 60\%$, we see the dramatical increase of ER. Specifically, when $\omega = 10\%$, ER becomes extremely large, i.e., $ER = 3.47e9$. However, when the percentage of ground-truth data in the input is sufficiently large (i.e., $\omega \geq 70\%$), ARIMA can perform very well and its performance is close to that of LSTM. This results strongly emphasizes the advantage of our model in predicting future traffic with only a small portion of ground-truth information.

3.6.6 Evaluating the performance of multi-step-ahead prediction

In this section, we conducted two experiments to evaluate the performance of the models in multi-step-ahead prediction ($T > 1$). In the first experiment, at every time-step, we predicted the traffic of the next 3 time-steps (equals to 15 minutes in the Abilene dataset) while letting the monitoring ratio vary from 10% to 90%. In the second experiment, we fixed the monitoring ratio at 70% but change the value of T to 6 and 9. The first experiment shows how precise the model works in different monitoring ratio while the second one is to see how well the model can predict the traffic in the far future. In each time-step, our proposed algorithm and LSTM predict the traffic matrices by applying the Iterated Multi-Step approach, while ARIMA uses the Direct Multi-Step approach [48]. As in the previous section, the results of ARIMA are extremely large and then, are presented in table 3.4.

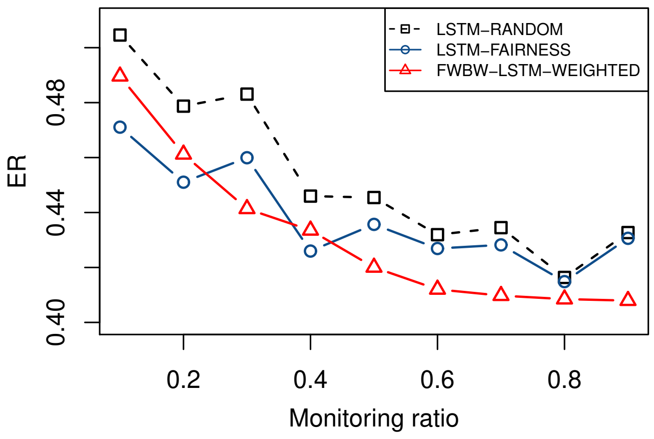


Figure 3.13: Performance comparison in multi-step-ahead prediction ($T = 3$).

Fig.3.13 shows the performance evaluation of all algorithms in terms of *ER*. In general, similar to the first experiment, our approach achieves the best performance in all scenarios, followed by LSTM and ARIMA. In comparison with LSTM, when $\omega = 90\%$, the *ER* of our algorithm is about 5.7% and 5.3% less than that of LSTM-RANDOM and LSTM-FAIRNESS. Similar to the first experiment, ARIMA also shows very poor performance compared to the other two approaches.

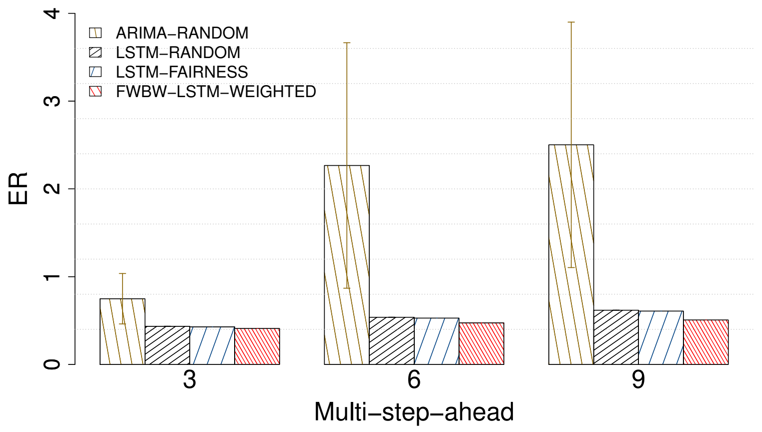


Figure 3.14: The Error Ratio of all models with different multi-step-ahead prediction.

Comparing the results of the first experiment with the one at section 3.6.5, it can be seen that the performance of the three algorithms in the multi-step-prediction is worse than that in the one-step-ahead prediction. Although our results in multi-step-ahead prediction are the best in most cases, the degradation when compared with the results of one-step-ahead prediction experiment is larger than that of ARIMA and LSTM. For example, in the case $\omega = 60\%$, the prediction errors of ARIMA, LSTM-RANDOM and LSTM-FAIRNESS increase by only 28.2%, 21.5% and 23.2%, respectively, while that of FWBW-LSTM-WEIGHTED is 54%.

Figure 3.14 shows the Error Ratio of all models with different values of T. In general, our model still shows the best performance in all cases. It is obvious that the higher the value of T, the higher error in the prediction. However, ARIMA has performed poorly by increasing 234% error in the case $T = 9$ compared to $T = 3$. LSTM-RANDOM, LSTM-FAIRNESS and FWBW-LSTM, on the other hand, only increase 42.3%, 42% and 23.5% in the same comparison, respectively.

3.6.7 Evaluating the effect of the input size (H)

The accuracy of statistical models such as ARIMA heavily depends on the amount of data fed to the model. We have seen in Section 3.6.5 and 3.6.6 the poor performance of ARIMA when only 30 previous data is used to predict the future value. To evaluate the effect of the input size (i.e., H), in this section, we present the comparison in the Error Ratio between ARIMA, LSTM-RANDOM, and FWBW-LSTM-RANDOM with different values of input size. Since one-day data of the Abilene dataset contains 288 time-steps, the experiments were conducted with H equals 36, 72, 144, and 288 which corresponds to 3-hour, 6-hour, 12-hour, and one-day of previous data is fed to the model, respectively. We fixed the monitoring ratio at 40%, and run the experiments 50 times for each model, and then plot the average results on Fig.3.15. Note that since the running time of ARIMA is too long, we only run the experiment 10 times for each scenario.

According to the results showed in Fig. 3.15, we observe two different trends in the results of the LSTM-based models (i.e., LSTM-RANDOM and FWBW-LSTM-RANDOM) and the ARIMA model. First, the performances of LSTM and our proposed model are steadily low regardless of the input size, although their figures make a small increase when the input size increases (at $H = 288$). This can be explained as the increase in the input size leads to the increase in the model complexity (e.g., the number of trainable variables in the neural network) which make the model hard to train.

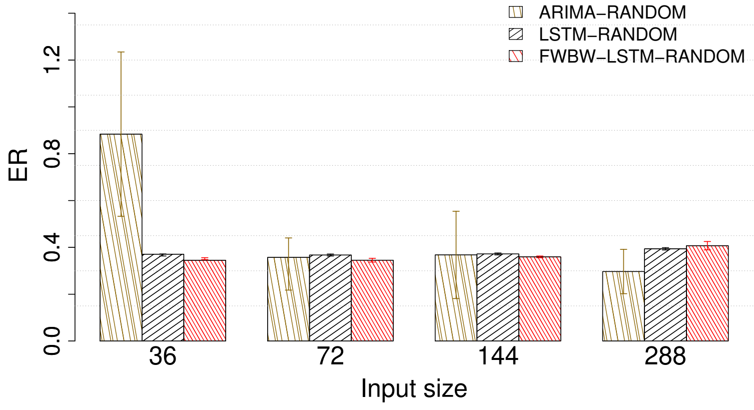


Figure 3.15: The effect of input size.

On the other hand, although the Error Ratio of ARIMA is considerably high at the low input size experiment ($H = 36$), its figure significantly drops in the next case when $H = 72$ (about 60%) and remains low in the others. Especially, with the input size equals to 288, the Error Ratio of ARIMA is 25% and 27% less than the LSTM and FWBW-LSTM. However, since ARIMA is a state space model, we need to fit the data to the model when we make a prediction for a flow at every time-step. Thus, in the experiments, the ARIMA takes a long running time doing the traffic prediction for all the flows at each time-step (about 64.83(s) and 143.13(s) in the cases $H = 36$ and $H = 288$ respectively). In contrast, Deep Learning-based models such as LSTM can be trained in offline mode and then, used for online traffic predicting.

3.6.8 Evaluating the time complexity

In this part, we evaluate the training and prediction time of the LSTM-based models (i.e., LSTM and FWBW-LSTM). Besides that, we also compare the models in terms of the number of trainable parameters and the occupied memory.

	LSTM	ConvLSTM	FWBW-LSTM
Training time (s)	435	**129**	778
Prediction time (s)	0.08	**0.03**	0.15

Table 3.5: The training and prediction time of LSTM and FWBW-LSTM.

	LSTM	ConvLSTM	FWBW-LSTM
Trainable parameters	**23425**	4465072	214944
Memory usage (MiB)	**147**	4467	237

Table 3.6: Comparison in the number of parameters and the amount of used memory.

In general, although having a long training time, LSTM, ConvLSTM, and FWBW-LSTM perform extremely fast in predicting traffic matrix at each time-step (less than a second). Therefore, LSTM-based models are more suitable for doing online traffic matrix prediction than the ARIMA model.

According to table 3.5 and 3.6, although being the most complicated model, ConvLSTM network has lowest training and prediction time. ConvLSTM can reduce the training and prediction time since it performs traffic prediction for the whole matrix as one (which means higher memory and computation usage) while LSTM-based models process the traffic flows separately.

Comparing the LSTM and FWBW-LSTM, while our model has nearly 10 times in the number of trainable parameters, the training and prediction time is only 2 times more than that of the LSTM model.

3.7 Summary

In this chapter, we address the traffic matrix prediction problem in backbone networks where the future traffic is estimated under the lack of precise historical data. We exploited the Convolutional LSTM network for extracting and modeling the spatiotemporal feature of the traffic matrices. We proposed a novel deep learning model and techniques which leverage the forward and backward ConvLSTM networks to correct input data and determine flows monitored at the next timestep. We conducted extensive experiments for evaluating our proposed approach. The results demonstrated that the proposed approach outperforms other well-known methods for time series analysis.

4 Network traffic imputation

Missing values appear in most multivariate time series, especially in the monitored network traffic data due to high measurement overhead and unavoidable loss. In the networking fields, missing data can prevent advanced analysis and downgrades downstream applications such as traffic engineering and anomaly detection. We propose GCRINT, a combination between Recurrent Neural Network (RNN) and Graph Convolutional Neural Network, for filling the missing values of network traffic data. We use a bidirectional Long Short-Term Memory network and Graph Neural Network to efficiently learn the spatial-temporal correlations in partially observed data.

4.1 Problem statement

In this work, we target the problem of recovering the missing values in the network traffic dataset. The imputation process will be done in the data preprocessing step before the prediction model is trained. The network model and the problem will be described as follows. The network is represented by a directed graph $G = (V, E)$, where

V is the set of nodes ($|V| = N$), and E is the set of the network's links. Each link $e \in E$ has capacity $c(e)$. The measurement of traffic data is the traffic volume exchanged between all the source-destination pairs in the network. Thus, the total number of traffic flows is $D = N^2$.

Let $X_t \in \mathbb{R}^D$ be the vector of all traffic volumes and $x_i^t \in X_t$ denote the traffic volume of flow i at the time-step t. We denote a partially measured data as $X = [X_1, X_2, ..., X_T](X \in \mathbb{R}^{T\times D})$, with T the total monitored time-steps. X is generally an incomplete matrix where each column $X^d = [x_1^d, ..., x_T^d]$ denotes the d-th time series and each row $X_t = [x_t^1, ..., x_t^D]$ is the data at time step t. We use a mask matrix $M \in \{0, 1\}^{T\times D}$ to indicate the locations of the missing value, where $m_t^d = 0$ if x_t^d is the missing value, and 1, otherwise. The imputation problem can be formulated as:

$$\begin{aligned} &\min_{\hat{X}} \sum_{t=1}^{T}\sum_{d=1}^{D} |\hat{x}_t^d - x_t^d| \\ s.t. &\sum_{t=1}^{T}\sum_{d=1}^{D} m_t^d * |\hat{x}_t^d - x_t^d| = 0 \end{aligned} \tag{4.1}$$

where $\hat{x}_t^d \in \hat{X}$ is the imputed data of X.

4.2 Graph convolutional recurrent neural network for imputing network traffic

We propose Graph Convolutional Recurrent Neural Network for Imputing Network Traffic (GCRINT), a spatial-temporal deep learning model for network traffic imputation. GCRINT is a combination of the RNN-based model and Graph-based neural network. We design a multi-layers model whose each layer has two modules for learning features in time and space domains. Like BRITS [7], we use a bidirectional LSTM model for learning the temporal feature in the traffic data. To cope with the dynamic correlation mentioned above, we use Graph Convolutional Neural Network to exploit the spatial feature among the traffic flows automatically. Then, each layer's outputs are combined by a fully connected layer to obtain the final imputed data. Furthermore, to address the LSTM model's scalability in handling long sequences, the input is reduced by half after each layer. In this way, we can still learn the long-range dependency on deeper layers while lowering model complexity.

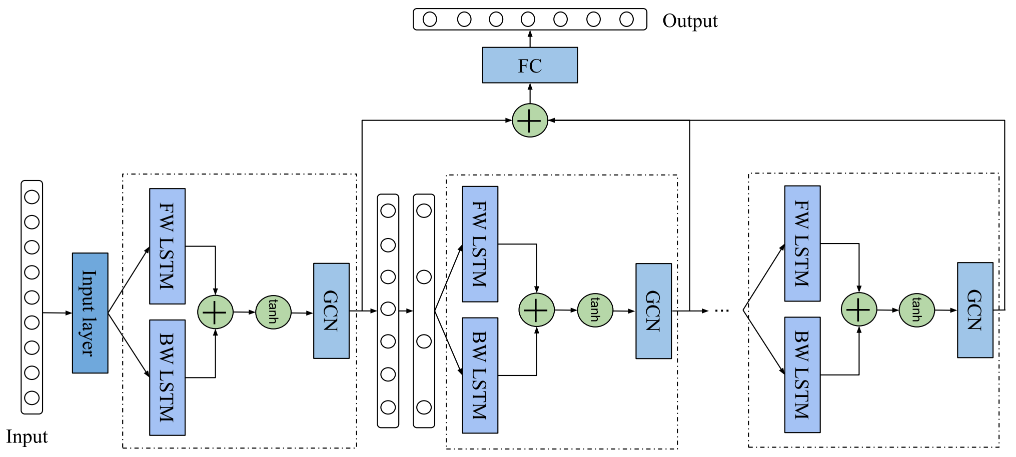


Figure 4.1: The architecture of GCRINT model.

Figure 4.1 describes the overall architecture of our proposed model. GCRINT is the combination of BiLSTM and GCN. Overall, the proposed model has three main modules: an input layer, BiLSTM, and GCN layers. The input layer is responsible for extracting hidden features from the input, while the BiLSTM and GCN layers are for learning the temporal and spatial features in the data. GCRINT has multiple layers of BiLSTM and GCN; thus, it can handle temporal and spatial dependencies at different levels. After each layer, the number of time-steps in the sequence data is reduced by half. By skipping some time-steps in the sequence data, we reduce the computational complexity in the latter layers while still learning the long-term temporal information. We combine the outputs from each layer and use a fully connected layer to obtain the final output. Next, we present the details of each module in GCRINT.

4.2.1 Input layer

In contrast to the tensor completion-based approaches [18, 64], which process the whole data at once, in the deep learning-based approaches, data is divided into sub-sequences and put into the imputation model sequentially. Let $[X_{1:T}, M_{1:T}]$ be the input of the GCRINT model in which $X_{1:T}$ is the traffic data of T time steps and $M_{1:T}$ is its corresponding mask matrix. Thus, the input of GCRINT is a 3D tensor $[T, D, 2]$ with two features: the traffic volume and the mask. In the input layer, we use two fully

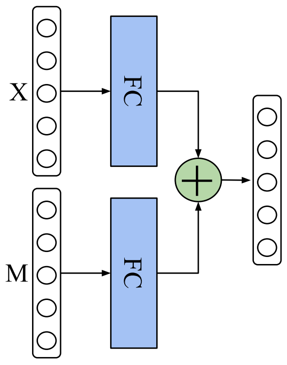


Figure 4.2: The design of the input layer.

connected networks to obtain the feature representation of the input (as shown in Fig. 4.2.1). After passing the input layer, we receive the output of $\mathcal{X}_{1:T}$, which is a 3D tensor with the size of $[T, D, H_I]$ (H_I is the number of hidden units of the fully connected network in the input layer). Then, $\mathcal{X}_{1:T}$ is fed into the BiLSTM layers.

4.2.2 Temporal feature learning with BiLSTM

Unlike the normal BiLSTM that receives the same input for both forward and backward LSTM networks, the BiLSTM layer in GCRINT takes inputs with different alignments for each LSTM propagation direction. Specifically, the forward LSTM takes $\mathcal{X}_{1:T-2}$ as input and obtains output $\mathcal{X}^f_{2:T-1}$ for time-steps from 2 to $T-1$. Similarly, $\mathcal{X}_{3:T}$ and $\mathcal{X}^b_{2:T-1}$ are the input and output of the backward LSTM. Therefore, from both LSTM networks, we obtain the output for the same time-steps $[2 : T-1]$. This technique was presented in [7] to overcome the backpropagation issue caused by the missing values in the data. The final output of BiLSTM layer $\mathcal{X}_G$ is the mean of $\mathcal{X}^f_{2:T-1}$ and $\mathcal{X}^b_{2:T-1}$ (Eq. 4.2).

$$\mathcal{X}_G = \tanh\left((\mathcal{X}^f_{2:T-1} + \mathcal{X}^b_{2:T-1})/2\right) \tag{4.2}$$

The output of the BiLSTM layer is a 3D tensor with the size of $[T-2, D, H_B]$, where H_B is the hidden size of the LSTM networks.

4.2.3 Spatial feature learning with GCN

The GCN in each layer falls into the spatial-based, node-level graph neural networks. Each node in the graph represents the traffic of a network flow. We use Diffusion Convolutional Neural Network to extract the spatial relationships among the traffic flows, as shown in Equation (4.3).

$$Z = \sum_{k=0}^{K} f(P^k \mathcal{X}_G W_k) \tag{4.3}$$

where $\mathcal{X}_G$ is the input of GCN, K is the number of diffusion steps, P^k is the power series of the transition matrix, $f(.)$ is the activation function, and W_k is the learnable parameters. The transition matrix is computed by $P = D^{-1}A$ where A is the adjacency matrix of the graph, D is the diagonal matrix of node degrees, $D_{ii} = \sum_j A_{ij}$.

However, there is no explicit graph representing the relations among the traffic flows in this work. Therefore, we adopt the Self-adaptive Adjacency Matrix from [59] to learn the input data's adjacency matrix. The Self-adaptive Adjacency Matrix is obtained by Equation (4.4) proposed in [59].

$$A_{adp} = Softmax(ReLU(E_1 E_2^T)) \tag{4.4}$$

where E_1, E_2 are the learnable parameters. From (4.3) and (4.4), the DCNN with Self-adaptive Adjacency Matrix is represented as:

$$Z = \sum_{k=0}^{K} f(A_{adp}^k \mathcal{X}_G W_k) \tag{4.5}$$

The output of the GCN is a 3D tensor $Z_{2:T-1}$ with size $[T-2, D, H_G]$ (H_G is the hidden size of the GCN). Finally, The outputs of the GCN in each layer are combined and fed into a fully connected layer to obtain the final output.

4.3 Performance evaluation

4.3.1 Datasets and baseline methods

We conduct experiments on two real network datasets: Brain, and Abilene, available at [36]. Each dataset is divided into three sets: 70% for training, 10% for validating, and 20% for testing. Note that, although the Brain network has 161 nodes in total, most of them are regional nodes. Therefore, we only consider the aggregated traffic from 9 backbone nodes in the Brain network. The experiment's results can be reproduced at [1].

The baseline methods are:

- **GCP** [18]: the CP decomposition-based tensor completion approach.
- **NTC** [61]: the network traffic recovery model combines deep learning model (3D-CNN) and CP decomposition-based approach.
- **BRITS** [7]: This model is based on a bidirectional recurrent network with GRU-I cell to impute time series. We do not compare with E^2GAN [30] because E^2GAN uses the same GRU-I cell, and their performances are relatively the same.

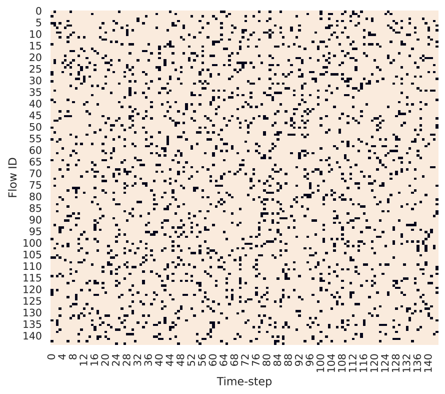
4.3.2 Performance metrics

We evaluate the imputation error using the Mean Absolute Error (MAE), which is calculated by Equation (4.6). For the traffic engineering problem, after obtaining the routing policy, the MLU is calculated using the actual traffic matrix from the test set.

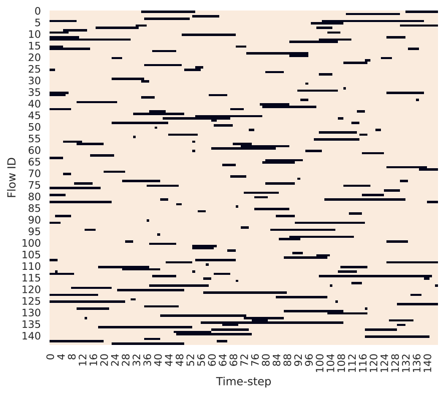
$$MAE = \frac{\sum_{t=1}^{N}\sum_{d=1}^{D}(1-m_t^d)|x_t^d - \hat{x}_t^d|}{\sum_{t=1}^{N}\sum_{d=1}^{D}(1-m_t^d)} \tag{4.6}$$

4.3.3 Generating synthetic missing data

To evaluate the imputation methods, we assume that the original datasets have no missing values. We synthetically generate the missing data by removing $k\%$ (i.e., the missing rate) entries from the original data following two missing scenarios. In the first scenario, the data is removed by two patterns: *random missing* and *block missing*



(a) Random missing.



(b) Block missing.

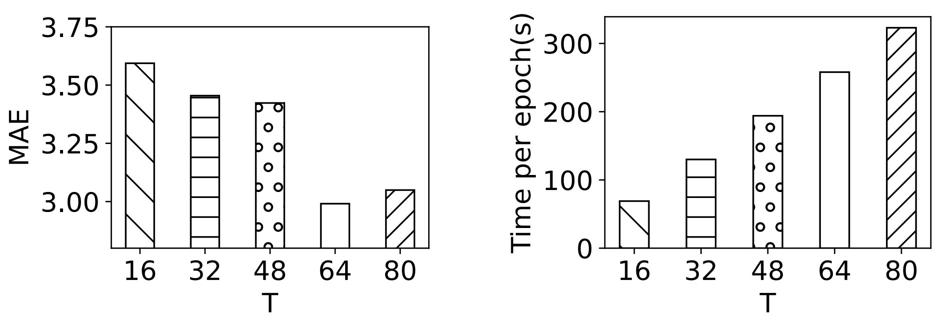
Figure 4.3: Example of random and block missing scenarios of Abilene dataset in 144 time-steps (12 hours). The black dots represent the missing values.

as showed in Fig.4.3. In the random missing, the entries are randomly removed, while in the block missing, a block of entries is removed consecutively. The value of k is varied from 50 to 90.

In the second scenario, we generate the missing data following the link failure events. In this scenario, we assume that the traffic flow is routed using the shortest path routing. When the link failure event occurs, all the measured data of the flows that pass through that link is not recorded. The link failure scenario is generated based on the study [20]. The details of this scenario will be described in Section 4.3.6.

4.3.4 Evaluating the impacts of the input size

The traffic dataset is divided into sub-sequences by using a sliding window of size T. Then, the missing data in each sub-sequence is recovered. Finally, the imputed data is obtained by taking the average of the overlapped recovered sub-sequences. In this experiment, we study the impacts of T on the imputation errors of GCRINT. Figure 4.4(a) shows the imputation errors of GCRINT on the Abilene dataset with 60% random missing values. As can be observed, the MAE decreases when the length of the input sequence (i.e., T) increases from 16 to 64 but the error increases beyond that. This phenomenon can be explained as follows. With a longer input sequence, the model



(a) Imputation errors.

(b) Training time per epoch.

Figure 4.4: GCRINT on different input sizes.

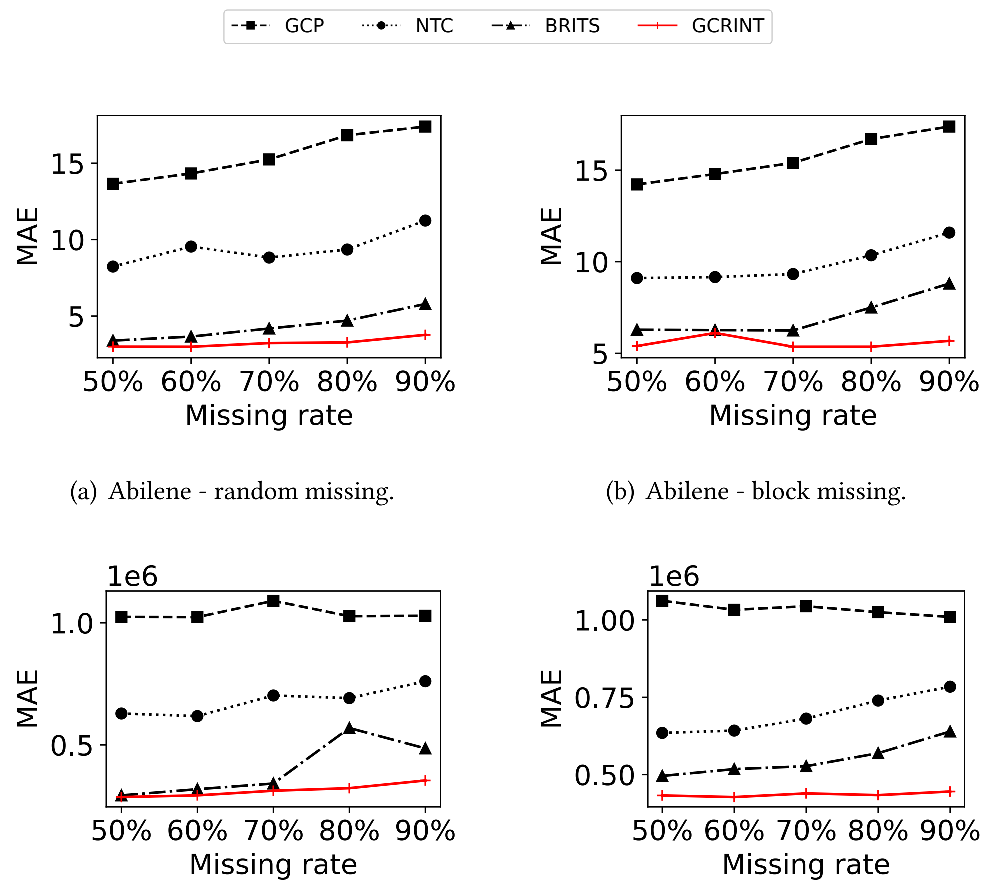
may receive more information to recover the missing values, thereby increasing the imputation accuracy. Using $T = 64$, we can reduce 16.8%, 13.4% and 12.6% in MAE compared to the cases $T = 16, 32, 48$, respectively. However, when the sequence length is sufficiently large ($T = 80$), the imputation model becomes too complicated, which leads to the LSTM network's inherent drawbacks in handling long sequences.

The impacts of T on the training time is depicted in Fig.4.4(b). There is a trade-off between imputation accuracy and computational complexity. The larger T, the more computational overhead in both model training and testing. Based on the experiment results obtained above, T should be set to a moderate value of 64. We use this value for all subsequent experiments.

4.3.5 Evaluating the imputation accuracy on random missing

Figure 4.5 shows the performance comparison of all the methods in terms of MAE with random and block missing scenarios. Overall, the LSTM-based models (i.e., GCRINT and BRITS) outperform the CP decomposition-based approaches (i.e., GCP and NTC). Our proposed model achieves the best performance in all the scenarios. All the methods have high imputation errors in the block missing in both datasets compared to the random missing scenario.

In comparison with GCP and NTC, GCRINT can reduce MAE by at most 80.5% and 68.7% on the Abilene dataset, and 72.0% and 54.4% on the Brain dataset. Comparing BRITS and GCRINT, while both models have almost the same MAE in the low missing



(a) Abilene - random missing.

(b) Abilene - block missing.

(c) Brain - random missing.

(d) Brain - block missing.

Figure 4.5: The comparison of all methods in terms of MAE with different missing scenarios.

rate (i.e., less than 70%), GCRINT achieves better performance when the missing rate increases. In the highest missing rate, GCRINT reduces MAE by about 35%.

4.3.6 Evaluating the imputation accuracy on missing data caused by link failures

In this experiment, we evaluate the performance of the proposed model in imputing missing data caused by link failures. We assume that the traffic flow is routed using the shortest path routing. When the link failure occurs, the measured data of the flows traveling on that link is missing. We generate the link failure scenarios based on the study in [20]. We use the Abilene dataset and compare the performance of GCRINT, BRITS, and NTC in this experiment.

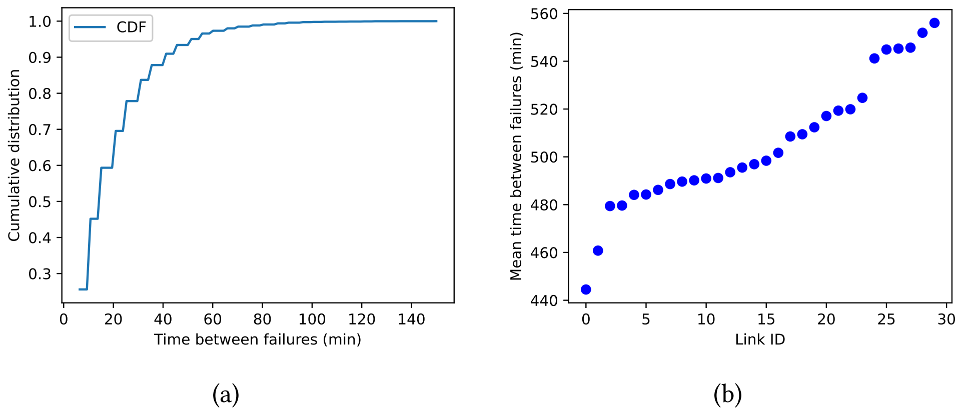


Figure 4.6: The cumulative distribution of times between failures over the entire network (a) and the mean time between failures per link (b).

According to the study in [20], a lot of failures happen close to each other. The mean time between two link failure events of a given link can be as low as a few minutes or as high as several days. In this experiment, we consider the minimum time between two failure events that happen in the entire network is 100 minutes, there are about 15 failure events that happen in one day. Since we use the Abilene dataset which the data was measured 288 times per day, the maximum percentage of link failure evens in one-day measurements is approximately 5%. Therefore, we generate the missing dataset with the percentage of link failure events that varies from 1% to 4%. Figure 4.6 shows the cumulative distribution of times between failures over the entire network and the mean time between failures per link when the percentage of link failure event equals 1% per day. The duration of the failure event is randomly set from 5 to 20 minutes. Table 4.1 shows the missing rate of the generated dataset corresponding to the percentage of link failure events. Figure 4.7 shows an example of missing values caused by link failures (1%) of Abilene dataset in 144 time-steps (12 hours).

Table 4.2 shows the performance comparison of GCRINT, BRITS, and NTC on recovering missing values caused by link failure events. The numbers in the parentheses are the performance improvements between GCRINT and other methods. We can

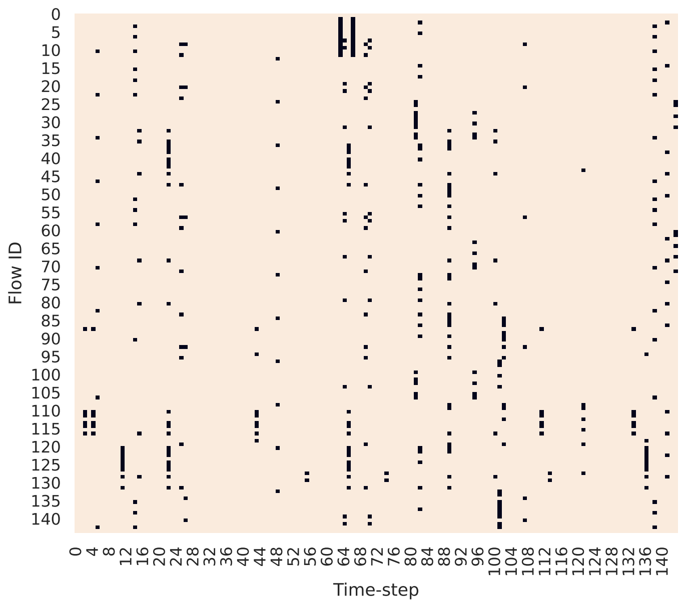


Figure 4.7: Example of missing values caused by link failures of Abilene dataset in 144 time-steps (12 hours). The black dots represent the missing values.

Percentage of link failures (%)	1	2	3	4
Missing rate (%)	2.28	4.49	6.7	8.76

Table 4.1: The missing rate of the generated datasets.

see that the time series completion-based models (GCRINT and BRITS) outperform the NTC approach. GCRINT can reduce the imputation error by 70% compared to the results of NTC. In the comparison between GCRINT and BRITS, GCRINT has better results in most cases. GCRINT reduces the error at most 5% (the case of 4% link failures) compared to BRITS. As shown in the table, the performance gaps between GCRINT and BRITS increase along with the increase of the link failure percentage. Since our model has the GCN module to utilize the spatial information in the data, in case of a high missing rate, GCRINT can achieve better performance than BRITS (which can only extract the temporal information).

Percentage of link failures (%)	1	2	3	4
GCRINT	1.55	**1.59**	**1.63**	**1.65**
BRITS	**1.53** (-1.4%)	1.61 (1.29%)	1.71 (4.34%)	1.72 (4.86%)
NTC	5.73 (73.02%)	5.89 (73.06%)	6.16 (73.54%)	6.50 (74.79%)

Table 4.2: Performance comparison of imputation methods on recovering missing values in network traffic dataset caused by link failure events.

4.3.7 Evaluating the impacts of imputed datasets on traffic prediction-based TE

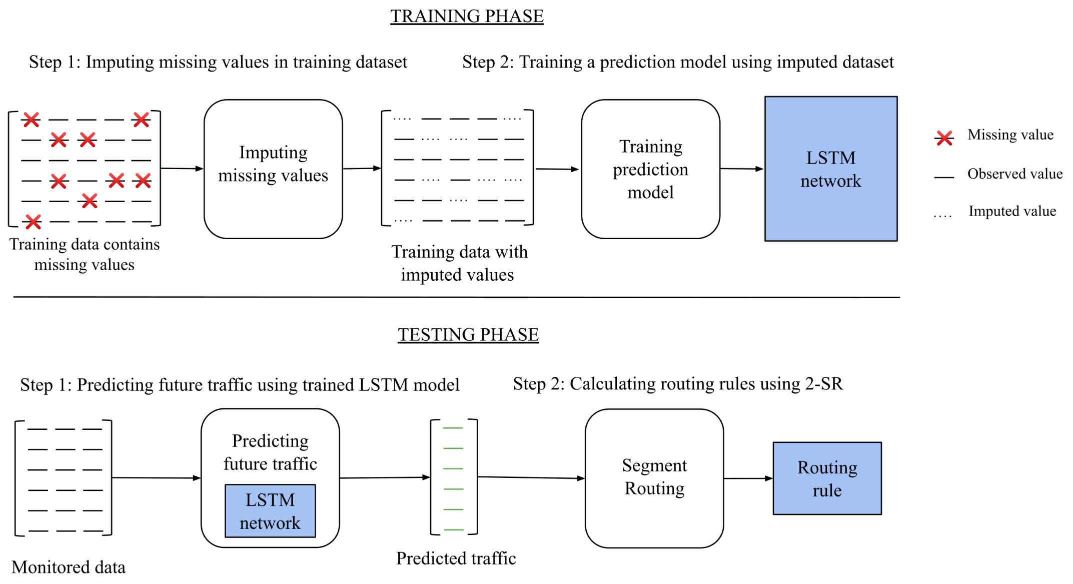

TRAINING PHASE
Step 1: Imputing missing values in training dataset
Step 2: Training a prediction model using imputed dataset
Imputing missing values
Training prediction model
LSTM network
Missing value
Observed value
Imputed value
Training data contains missing values
Training data with imputed values
TESTING PHASE
Step 1: Predicting future traffic using trained LSTM model
Step 2: Calculating routing rules using 2-SR
Predicting future traffic
LSTM network
Segment Routing
Routing rule
Monitored data
Predicted traffic

Figure 4.8: Traffic Engineering leveraging network traffic imputation.

In this experiment, we evaluate the impact of the imputation methods on traffic prediction-based TE. There are two separate phases in this experiment: the training phase and the testing phase (see Figure 4.8). In the training phase, we use imputation methods (e.g., GCRINT, BRITS, NTC, GCP) to recover the missing values in the training dataset. After estimating all the missing data, the imputed dataset is used to train a prediction model (e.g., LSTM network). In the testing phase, the network controller obtains the future traffic demands using the LSTM network and monitored data. Then, routing rules are calculated using the 2-segment routing algorithm [5] based on the predicted values. The maximum link utilization (Equation 4.7) is used to evaluate the performance of the traffic engineering task. E, $load(e)$, and $c(e)$ are the set of network link, the traffic load on link e, and the capacity of link e, respectively.

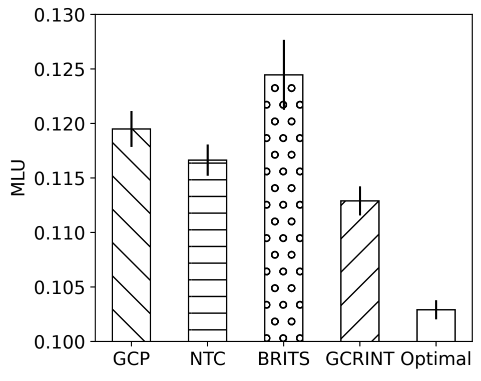
$$MLU = \max_{e \in E} \frac{load(e)}{c(e)} \tag{4.7}$$

In the training phase, the missing rate of the training dataset is set at 50%. The final results (ie., MLU) are named based on the imputation methods used in the training phase. In addition, we also train the LSTM network using the data without missing values (i.e., ground-truth data). The results of this case are named 'Optimal'. By using the same prediction model and routing algorithm, we can evaluate the impact of the different imputation methods on the traffic-prediction-based TE.

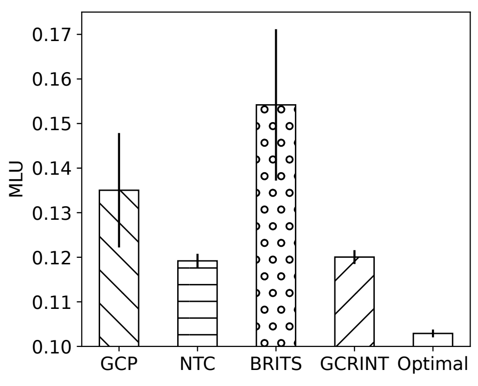
Overall, traffic engineering results reflect the imputation methods' performance, as shown in Fig.4.9. The average MLU of GCRINT is 22% lower than BRITS in the block missing scenario of the Abilene dataset. In the Brain dataset, GCRINT also reduces the MLU by about 70% to 80% on average, compared with that of GCP and NTC.

Although the average MLU of all the methods is close to the optimal, GCP and NTC suffer from a significantly high MLU in some absurd time-steps, especially in the Brain dataset. Therefore, the variances of GCP and NTC are considerably larger than those of GCRINT. Similarly, solving routing using imputed data by BRITS leads to high link utilization in many time-steps of the Abilene dataset (Fig.4.9(a), 4.9(b)). In most cases, GCRINT achieves the lowest MLU.

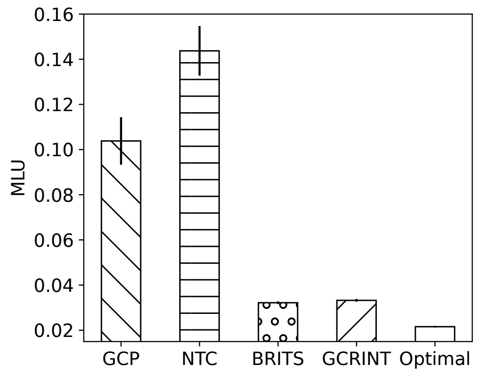
In conclusion, the recovered data provided by our proposed model facilitates the most stable performance in traffic engineering compared with all recently proposed methods.



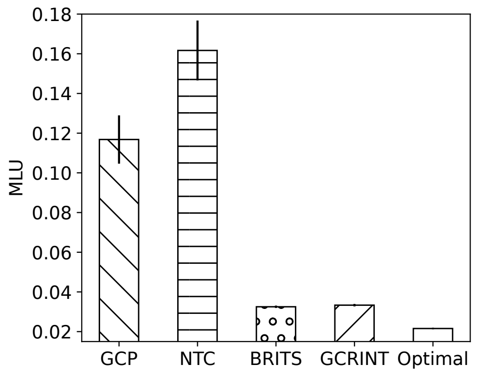
(a) Abilene - random missing.



(b) Abilene - block missing.



(c) Brain - random missing.



(d) Brain - block missing.

Figure 4.9: The comparison of imputation methods in traffic prediction-based TE.

4.4 Summary

In this chapter, we present the novel model, namely GCRINT, to address the network traffic imputation problem. To exploit the unique characteristic of traffic data, GCRINT is constructed with three main modules for extracting temporal and spatial features. Extensive experiments demonstrated our model's effectiveness in real network traffic datasets (e.g., Brain and Abilene datasets) with different missing scenarios. Moreover, we showed that GCRINT can improve the performance of downstream network applications such as traffic engineering.

5 Traffic prediction-based traffic engineering

Based on the concept of source routing, segment routing (SR) allows the source or ingress node to inject a sequence of segment labels into the packet header and specify the routing path. Due to the routing flexibility, SR has been widely used to solve traffic engineering (TE) problems such as minimizing the maximum link utilization of a network. However, most of the prior studies only solve the problems in a single snapshot without considering network traffic dynamics, resulting in frequent traffic reroutes. Furthermore, the traditional TE solutions usually require a large amount of traffic data, leading to high traffic monitoring overhead.

To cope with these issues, we focus on solving the segment routing-based traffic engineering problems by taking into account future traffic changes. In Section 5.1, we present the TE problem with 2-segment routing (2SR). Then, we formulate the multi-time-step segment routing problem (MTSR) with traffic prediction in Section 5.2. Section 5.3 gives details about our proposed heuristic to solve MTSR problem. In addition in Section 5.4, we present a combined approach of MTSR and compressive sensing for reducing the monitoring cost. The performance evaluation and summary are showed in Section 5.5 and 5.6, respectively.

5.1 Problem statement

First, we briefly introduce the TE problem with 2-SR. This problem was introduced in [5] as a traffic matrix aware segment routing. Therefore, some notations and figures from [5] are reused in this dissertation. However, in contrast with their problem, we do not consider that the traffic flows can be arbitrarily split and routed by different paths.

The network is represented by an undirected graph $G = (V, E)$, where V is the set of nodes ($|V| = N$), and E is the set of the network's links. Each link $e \in E$ has a capacity $c(e)$. Let $M_t \in \mathbb{R}^{N\times N}$ be the traffic matrix at time-step t and $m^t_{ij} \in M_t$ denote the traffic flow from node i to j (flow ij for short, $i, j \in V$) at time step t. Let α^k_{ij} be the binary variable which $\alpha^k_{ij} = 1$ indicates that flow ij is routed through intermediate node k, and otherwise. Therefore α^k_{ij} can be considered as the routing policy. We assume that the network is controlled by a central controller such as SDN controller [12]. The controller plays an important role in collecting the knowledge of the network (i.e., topology, traffic statistics), predicting future traffic demands, and applying the routing policy to devices via PCEP [49]. However, the implementation of the controller is out-of-scope and will be omitted.

In 2-SR, we only need to select one intermediate node k for each flow ij. Figure 5.1 shows the example of 2-segment routing path for flow ij with the intermediate node k. The traffic from i to k and from k to j is routed through the shortest path between them. The intermediate node $k = i$ or $k = j$ means that flow ij is routed by the shortest path from i to j. The problem (called P_0) can be formulated as the following integer linear program. The variable θ represents the maximum link utilization.

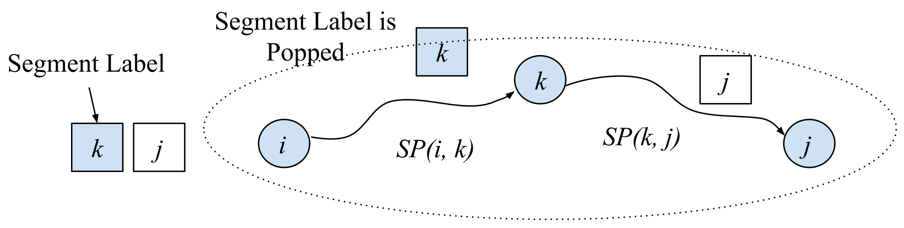


Figure 5.1: Illustration of 2-segment routing [5].

P_0 :

$$\text{minimize} \quad \theta \tag{5.1}$$

$$\sum_{k \in V} \alpha_{ij}^k = 1 \quad \forall i, j \in V \tag{5.2}$$

$$\sum_{ij} \sum_{k} g_{ij}^k(e) \alpha_{ij}^k m_{ij}^t \leq \theta c(e) \quad \forall e \in E \tag{5.3}$$

$$\alpha_{ij}^k \in \{0, 1\} \quad \forall i, j, k \in V \tag{5.4}$$

We have $g_{ij}^k(e) = f_{ik}(e) + f_{kj}(e)$, in which $f_{ik}(e) = 1$ if link e is on the shortest path from i to k of flow ij with intermediate node k and $f_{ik}(e) = 0$, otherwise. Note that in P_0, the maximum link utilization θ can be greater than 1, which means the network may be congested. Equations (2) and (4) ensure that all traffic from i to j is routed and cannot be split into different paths. Equation (3) depicts that the total traffic load on link e is not greater than the link's capacity.

By solving the problem above, we can get a routing policy for a single time-step t. Although we can also reuse the same routing policy for the next T time-step to mitigate the routing path's variation, the link utilization constraint may not be assured due to network traffic's dynamic behavior. Therefore, for future time-steps, we need to resolve the problem and update the routing policy. A trivial approach to minimize the maximum link utilization in the next T time-steps is to solve the problem P_0 at every time-step. This approach may lead to a considerable number of re-routed flows. To this end, in the next section, we propose an extension of P_0, which addresses the segment routing problem in multiple steps. We first present the mathematical formulation and then describe some theoretical analysis.

Problem	Routing cycle	Required number of predicted traffic matrices	Problem complexity
P_0	1 time-step	No traffic prediction	Low
P_1	T time-steps	T traffic matrices of every time-step in the next cycle	High
P_2	T time-steps	One traffic matrix of maximum demands in the next cycle	Low
P_3	T time-steps	P matrices of maximum demands of every sub-periods in the next cycle	Medium

Table 5.1: The differences of the problem formulations.

5.2 Multi-time-step Segment Routing

5.2.1 Problem formulations

In this section, we present the formulation of the multi-time-step segment routing problem (called P_1) and its modified versions (P_2 and P_3). As mentioned in Section 5.1, by solving these problems, we obtain routing policies that can be applied for the next routing cycle, which is T ($T > 1$) time-steps ahead. Here T is the length of the routing cycle in the number of time-steps.

Assume that $M = [M_1, M_2, ..., M_T]$ are the predicted traffic matrices of the next T time-steps, which can be acquired by using the state-of-the-art prediction models such as in [26], [59]. We extend the problem P_0 for T future steps by considering more constraints corresponding to the traffic demands of each time-step.

P_1 :

$$\text{minimize} \quad \theta \tag{5.5}$$

$$\sum_{k\in V} \alpha_{ij}^k = 1 \quad \forall i, j \in V \tag{5.6}$$

$$\sum_{ij}\sum_{k} g_{ij}^k(e)\alpha_{ij}^k m_{ij}^t \leq \theta c(e) \quad \forall e \in E; \forall t \in T \tag{5.7}$$

$$\alpha_{ij}^k \in \{0, 1\} \quad \forall i, j, k \in V \tag{5.8}$$

It can be seen that P_1 is different from P_0 by the link capacity constraints (7). Specifically, in P_1, the routing policy α_{ij}^k needs to satisfy link capacity constraints at “every" time-step. Due to a larger number of constraints, problem P_1 becomes more complicated than P_0. Besides that, P_1 requires the prediction of all traffic matrices of the next T time-steps, which remains a significant challenge even with the most state-of-the-art deep learning models. Obviously, prediction accuracy plays a vital role in the performance of problem P_1. Unfortunately, the results of some proposed prediction models [26], [59] show that the prediction performance worsens when the number of prediction steps increases. Therefore, to reduce the problem complexity and alleviate the burden of traffic prediction tasks, we formulate problems P_2 and P_3, which are the relaxed versions of P_1.

P_2 :

$$\begin{aligned} \text{minimize} \quad & \theta && (5.9) \\ \sum_{k \in V} \alpha_{ij}^k &= 1 \quad \forall i, j \in V && (5.10) \\ \sum_{ij} \sum_k g_{ij}^k(e) \alpha_{ij}^k \max_{t \in T} m_{ij}^t &\leq \theta c(e) \quad \forall e \in E && (5.11) \\ \alpha_{ij}^k &\in \{0, 1\} \quad \forall i, j, k \in V && (5.12) \end{aligned}$$

In problem P_2, we only consider the maximum values of each flow ij over the next T time-steps. By doing so, P_2 has the same number of constraints as P_0. Besides, we only need to predict one traffic matrix whose elements are the maximum value of each traffic flow.

P_3 :

$$\begin{aligned} \text{minimize} \quad & \theta && (5.13) \\ \sum_{k \in V} \alpha_{ij}^k &= 1 \quad \forall i, j \in V && (5.14) \\ \sum_{ij} \sum_k g_{ij}^k(e) \alpha_{ij}^k \max_{t \in T_p} m_{ij}^t &\leq \theta c(e) \quad \forall e \in E; \forall T_p && (5.15) \\ \alpha_{ij}^k &\in \{0, 1\} \quad \forall i, j, k \in V && (5.16) \end{aligned}$$

In problem P_3, the routing cycle is divided into P sub-periods in which each of them has T_p time-steps. Then, similar to P_2, we formulate the problem with the maximum values of flow ij in each sub-period T_p. Accordingly, to solve P_3, we only need to predict P traffic matrices, which are the maximum traffic of every flow ij in each sub-period T_p.

The differences in the proposed problem formulations are summarized in table 5.1. In general, the required number of predicted traffic matrices depends on the way we estimate future traffic.

5.2.2 Theoretical analysis

In this part, we theoretically analyze the performance ratios of P_2, and P_3 to P_1. Denote (θ_1^*, α_1^*), (θ_2^*, α_2^*), and (θ_3^*, α_3^*) as the optimal solutions obtained by solving P_1, P_2, and P_3, respectively. We denote $u(t, e, \alpha_p^*)$ the utilization of link e when we apply the routing policy α_p^* in routing cycle t. Then, the maximum link utilization of the network when applying the routing policy (α_p^*), denoted as $u(\alpha_p^*)$, is defined as follows:

$$u(\alpha_p^*) = \max_{\forall t,e} u(t, e, \alpha_p^*) = \max_{\forall t,e} \frac{\sum_{ij} \sum_k g_{ij}^k(e)(\alpha_p^*)_{ij}^k m_{ij}^t}{c(e)}$$

Theorem 5.1

- $\theta_1^* = u(\alpha_1^*); \theta_2^* \geq u(\alpha_2^*); \theta_3^* \geq u(\alpha_3^*)$
- $\theta_1^* \leq \theta_3^* \leq \theta_2^*$

Proof. According to (5.7), we have:

$$\theta_1^* \geq \frac{\sum_{ij} \sum_k g_{ij}^k(e)(\alpha_1^*)_{ij}^k m_{ij}^t}{c(e)} = u(t, e, \alpha_p^*) \ \ \forall t, e$$

As θ_1^* is the optimal solution of P_1, the following equation holds:

$$\theta_1^* = \max_{\forall t,e} \frac{\sum_{ij} \sum_k g_{ij}^k(e)(\alpha_1^*)_{ij}^k m_{ij}^{t_1}}{c(e)} = u(\alpha_1^*)$$

Concerning P_2, we have:

$$\begin{aligned} \theta_2^* &\geq \frac{\sum_{ij} \sum_k g_{ij}^k(e)(\alpha_2^*)_{ij}^k \max_t m_{ij}^t}{c(e)} \ \ \forall t, e \\ &\geq \frac{\sum_{ij} \sum_k g_{ij}^k(e)(\alpha_2^*)_{ij}^k m_{ij}^t}{c(e)} \ \ \forall t, e \\ \Rightarrow \theta_2^* &\geq \max_{\forall t,e} \frac{\sum_{ij} \sum_k g_{ij}^k(e)(\alpha_2^*)_{ij}^k m_{ij}^{t_2}}{c(e)} = u(\alpha_2^*) \end{aligned}$$

Similarly, we have: $\theta_3^* \geq u(\alpha_3^*)$. Now, we are going to prove that $\theta_1^* \leq \theta_3^* \leq \theta_2^*$.

First, we will prove the following hypothesis: *"If (α_3, θ_3) is a feasible solution of P_3, then it is also a feasible solution of P_1; if (α_2, θ_2) is a feasible solution of P_2, then it is also a feasible solution of P_3"*. According to this hypothesis, we derive that $\theta_1^* \leq \theta_3^*$ and $\theta_3^* \leq \theta_2^*$.
According to (5.15), we have:

$$\sum_{ij}\sum_{k} g_{ij}^k(e)(\alpha_3)_{ij}^k \max_{t\in T_p} m_{ij}^t \leq \theta_3 c(e) \ \ \forall T_p, e \tag{5.17}$$

As $\max_{t\in T_p} m_{ij}^t \geq m_{ij}^t$ ($\forall t \in T_p$), from (5.17), we can derive that

$$\sum_{ij}\sum_{k} g_{ij}^k(e)(\alpha_3)_{ij}^k m_{ij}^t \leq \theta_3 c(e) \ \ \forall t, e \tag{5.18}$$

It means that (α_3, θ_3) satisfies constraint (5.7), thus it is a feasible solution of P_1.
Similarly, according to (5.11), we have:

$$\sum_{ij}\sum_{k} g_{ij}^k(e)(\alpha_2)_{ij}^k \max_{t\in T} m_{ij}^t \leq \theta_2 c(e) \ \ \forall e \tag{5.19}$$

Let T_p is an arbitrary sub-period of T, then $\max_{t\in T_p} m_{ij}^t \leq \max_{t\in T} m_{ij}^t$. Therefore, from (5.19), we have:

$$\sum_{ij}\sum_{k} g_{ij}^k(e)(\alpha_2)_{ij}^k \max_{t\in T_p} m_{ij}^t \leq \theta_2 c(e) \ \ \forall T_p, e \tag{5.20}$$

It means that (α_2, θ_2) satisfies constraint (5.15), thus it is a feasible solution of P_3 □

Remark 1 *According to Theorem 5.1, when applying the routing policies obtained from solving the MTSR problem, the actual maximum link utilization of the network is less than its theoretical θ^*. In addition, since the performance of P_3 is bounded by P_1 and P_2 ($\theta_1^* \leq \theta_3^* \leq \theta_2^*$), we will derive the upper bound of $\frac{\theta_2^*}{\theta_1^*}$ which will also be the upper bound of $\frac{\theta_3^*}{\theta_1^*}$.*

In the following, we will analyze the performance ratio of P_2 and P_3 to P_1.

Theorem 5.2 *Let e^* be the link with the largest capacity, and e_2^* be the link where the equal sign of constraint (5.11) in P_2 holds; Then, the performance ratio of P_2 to P_1 is upper* *bounded by λ, which is calculated as follows.*

$$\lambda = \frac{\sum_{ij} \max_t m_{ij}^t}{\max_t \max_{ij} m_{ij}^t} \frac{c(e^*)}{c(e_2^*)} \tag{5.21}$$

Proof. According to (5.11), we have

$$\sum_{ij} \sum_{k} g_{ij}^k(e_2^*)(\alpha_2^*)_{ij}^k \max_t m_{ij}^t = \theta_2^* c(e_2^*) \tag{5.22}$$

We also have

$$\sum_{ij} \sum_{k} g_{ij}^k(e_2^*)(\alpha_2^*)_{ij}^k \max_t m_{ij}^t \leq \sum_{ij} \max_t m_{ij}^t \tag{5.23}$$

Therefore, from (5.22) and (5.23), we deduce that

$$\theta_2^* c(e_2^*) \leq \sum_{ij} \max_t m_{ij}^t \tag{5.24}$$

Concerning P_1, from constraint (5.7) and the assumption that e^* has the largest capacity, we have

$$\theta_1^* c(e^*) \geq \theta_1^* c(e) \geq \sum_{ij} \sum_{k} g_{ij}^k(e) \alpha_{ij}^k m_{ij}^t \quad \forall t; \forall e \tag{5.25}$$

As $\sum_{ij} \sum_k g_{ij}^k(e) \alpha_{ij}^k m_{ij}^t$ is the total amount of traffic routed through link e at time-step t, it should be greater than or equal to the traffic of any pair ij routed through e, it means that

$$\sum_{ij} \sum_{k} g_{ij}^k(e) \alpha_{ij}^k m_{ij}^t \geq \max_{ij} m_{ij}^t \tag{5.26}$$

(5.26) holds for all time-steps t. Therefore, from (5.25) and (5.26), we deduce that

$$\theta_1^* c(e^*) \geq \max_t \max_{ij} m_{ij}^t \tag{5.27}$$

Finally, from (5.24) and (5.27), we have

$$\frac{\theta_2^*}{\theta_1^*} \leq \frac{\sum_{ij} \max_t m_{ij}^t c(e^*)}{\max_t \max_{ij} m_{ij}^t c(e_2^*)} \tag{5.28}$$

□

Remark 2 *Assuming that all the network links have the same capacity, the performance ratio λ largely depends on the current network situation in the routing cycle. If all flows in the network have similar demands and are stable within the routing cycle, the value of λ could be much greater than 1. However, since most of flows in the network are mice flows and the network traffic is extremely dynamic, we have $\sum_{ij} \max_t m_{ij}^t \approx \max_t \max_{ij} m_{ij}^t$, and then $\lambda \approx 1$.*

According to Theorem 5.2, by solving the MTSR problem with formulation P_2, we can significantly reduce the problem complexity while still achieving good performance for the routing policy. In addition, P_2 only requires the predicted values of the maximum demand instead of the predicted demands of every time-steps in the next routing cycle. By doing so, P_2 alleviates the difficulty in the traffic prediction task. Therefore, we formulate the MTSR problem using the P_2 formulation. In the remaining part of this chapter, when mentioning the MTSR without any specification, we mean the MTSR with formulation P_2.

5.2.3 Traffic prediction for multi-time-step segment routing

As mentioned in the previous section, accurately predicted traffic matrices are required as input for all the problem formulations, except in the case we don't consider traffic prediction for P_0, i.e., using current traffic for TE. Note that, our objective is not to develop a new prediction model but to leverage the existing models for addressing the traffic engineering problem. There is a large number of proposed models for traffic matrix prediction such as in [4], [26]. Here we adopt the prediction model to meet the requirements of our problem formulations. For example, in problem P_1, at the beginning of each routing cycle, the prediction model uses the historical data of the last H time-steps to predict the traffic of the next T time-steps. In P_2, the prediction model only needs to infer the maximum demand for each traffic flow.

In this study, we use Graph WaveNet (GWN) [59] as the prediction model. Motivated by [35], GWN adopts stacked dilated casual convolutions to extract temporal features in the historical traffic data. In addition, GWN uses a Graph Convolutional Neural network with a self-adaptive adjacency matrix module which can exploit the spatial relations among traffic flows from training data. By combining these techniques, GWN can handle spatial-temporal data (e.g., traffic flows) and achieve better prediction accuracy than other time-series models such as Long Short-Term Memory (LSTM) and Auto-Regressive Integrated Moving Average (ARIMA). In this work, we apply GWN for traffic prediction with minor modifications, and the details of the model are out of the scope and will be omitted. The implementation and the details of the model training/testing processes for traffic prediction used in this work can be found at [2].

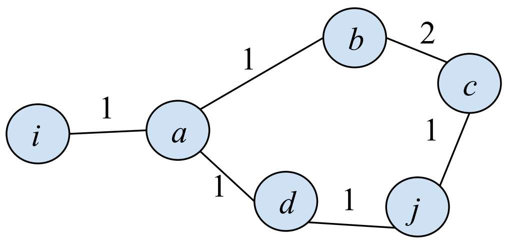
5.3 Local search algorithm for solving MTSR problem

The MTSR problem described in Section 5.2.1 is an integer programming problem with a vast search space. Traditional MILP solvers are hardly scaled to large topologies with more than 20 nodes [14]. Thus, using heuristic or meta-heuristic algorithms would be the most practical way to solve this problem rapidly. Gay, Hartert, and Vissicchio proposed a local search algorithm to solve the n-segment routing algorithm. We adapt the algorithm proposed in [14] to propose a new algorithm that effectively solves the multi-step segment routing problem by exploiting the structure of 2-segment routing called Local Search 2 Segment Routing (LS2SR). We also improve LS2SR on solving the MTSR problem by adding a mechanism to refine the routing policy from the previous cycle, thereby lessening the routing policy variation over different cycles.

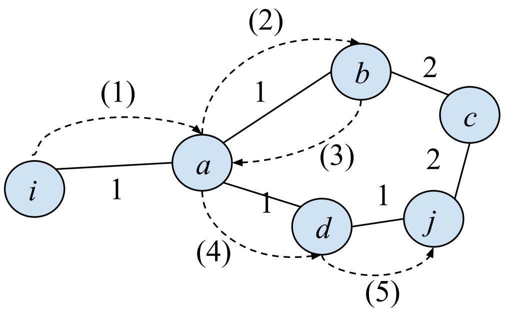
5.3.1 Search space reduction techniques

We begin by investigating the search space of the MTSR problem. In Section 5.2.1, we have mathematically formulated the MTSR problem by using binary variables α_{ij}^{k} which indicates whether a traffic flow from a source node i to a destination node j goes through an intermediate node k. These formulations contain two major sources of redundancy, namely equivalent paths and non-simple paths.

Figure 5.2(b) shows examples of equivalence paths. Consider a flow ij with a routing path $i \longrightarrow a \longrightarrow d \longrightarrow j$ going through two intermediate nodes a and d. In the formulations provided in Section 5.2.1, the path $i \longrightarrow a \longrightarrow d \longrightarrow j$ is duplicated in the search space as it is counted as a routing path going through a (i.e., $\alpha_{ij}^{a} = 1$), and also as a routing path going through d (i.e., $\alpha_{ij}^{d} = 1$).



(a) Equivalent paths.



(b) Non simple paths.

Figure 5.2: Examples of redundant paths in 2-segment routing.

The second type of redundancy is caused by so-called non-simple paths, i.e., paths that visit a link more than once. Figure 5.2 illustrates an example of non-simple path. In this example, the routing path from i to j with intermediate node b: $i \longrightarrow a \longrightarrow b \longrightarrow a \longrightarrow d \longrightarrow j$ (represented by dotted line) visits link (a, b) twice. Obviously, by pruning the looped route $a \longrightarrow b \longrightarrow a$ we obtain more efficient path $i \longrightarrow a \longrightarrow d \longrightarrow j$. Therefore, including such non-simple paths in the search space only causes redundant time complexity but can not improve the solution's goodness. It is obvious that the route constructed by middle point a will cost less than that of middle point b.

In the following, we propose a method to construct the routing path search space. Let $\mathcal{P}$ be the set of 2-segment routing paths of all the source and destination pairs, and $\mathcal{P}_{ij} \in \mathcal{P}$ denotes the set of 2-segment routing paths from node i to j. We construct $\mathcal{P}$ as follows. First, we enumerate all possible 2-segment paths from node i to j (with all possible intermediate nodes k). Then, we remove from $\mathcal{P}_{ij}$ ($\forall i, j \in V$) all the redundant paths defined as above. To reduce the search space, instead of using binary variables α_{ij}^{k}, we use integer variables x_{ij} which indicates the index of a routing path in $\mathcal{P}_{ij}$, i.e., $x_{ij} \in \{0, 1, ..., |\mathcal{P}_{ij}|\}$. Finally, a routing policy at routing cycle c is defined by the combination of all x_{ij} $\forall i, j \in V$, denoted as $x^c = [x_{ij}]$.

In the following, we first present the link and flow selection strategy in Section 5.3.2 and then describe the details of LS2SR in Section 5.3.3.

5.3.2 Link and flow selection strategy

LS2SR falls into the category of improvement-based heuristic algorithms, which starts from a feasible solution and improves the solution by applying successive changes. In particular, the initial solution of LS2SR is the shortest path routing. Then, at each iteration, LS2SR selects a flow that is crucial to minimize the maximum link utilization and alters its path. Motivated by the algorithm proposed in [14], we design a critical flow selection strategy in which the link with a higher traffic load will have a higher chance of being selected. Specifically, a link e is selected with a probability p_e defined as

$$p_e = \frac{load(e)^\beta}{\sum_{e \in E} load(e)^\beta} \tag{5.29}$$

where $\beta \in [0, +\infty)$ is the *intensification coefficient*. If β is high, then the selection distribution will be short-tailed. The link selection method will select a small subset of the highly loaded edges. Otherwise, the selection becomes more diverse. The reason behind this heuristic is that changing the path of flow passing through the highly loaded link will reduce the load in that link, thereby reducing the maximum link utilization of the whole network.

Subsequently, we select a flow passing through the selected link e, by the probability $p_{i,j}$:

$$p_{i,j} = \frac{(m_{ij})^\gamma}{\sum_{ij \in \mathcal{D}(e)} (m_{ij})^\gamma} \tag{5.30}$$

where $\mathcal{D}(e)$ denotes the set of flows routed on link e, and γ is the *intensification coefficient* for this selection distribution. Therefore, the large flow routed through the highly loaded link e is rerouted.

5.3.3 Local search for multi-time-step segment routing

In this part, we present the details of LS2SR. To alleviate the routing path change, in LS2SR, the routing path in the first cycle is initialized by the shortest path; In each subsequent cycle, the initial routing policy is the routing policy obtained from the previous cycle. The details of LS2SR are presented in Algorithm 2. We first sort the paths in $\mathcal{P}_{ij}$ by the increasing order of their cost (the sum of all the link costs in the path) and calculate the maximum link utilization corresponding to the initial solution

```
Algorithm 2: LS2SR Algorithm
   input : network $G(V,E)$; traffic matrix $M$; initial solution $x^{c-1}$,
           set of all the routing paths $\mathcal{P}$
   output: Routing path $x^{*c}$
 1 $\mathcal{P} = [sort(\mathcal{P}_{ij})] \quad \forall i,j \in V$ $x^{*c} = x^{c-1}$ $\theta^* = MLU(G, x^{*c}, M)$
 2 while not timeout do
 3     $e = select\_link(G, M, x^{*c})$ $i,j = select\_flow(G, M, e, x^{*c})$
        $x^c = select\_path(x^{*c}, \mathcal{P}_{ij})$ $\theta = MLU(G, x^c, M)$
 4     if $\theta^* - \theta > \epsilon$ then
 5         $x^{*c} = x^c$; $\theta^* = \theta$;
 6     end
 7 end
 8 return $x^{*c}$
```

(line 4). The loop from lines 5 to 12 is to search for a new routing solution that may reduce the maximum link utilization. In each iteration, we first select the link e and flow ij based on the selection strategies described in Section 5.3.2. Then, we choose a new path from $\mathcal{P}_{ij}$ for flow ij (line 8). The new path is chosen systematically as follows: as $\mathcal{P}_{ij}$ has been sorted by the paths' cost, we select the path that has the next higher cost than the current path. The rationale behind this path selection operator is to avoid the highly loaded link while trying not to select the path with a significantly high cost. If the maximum link utilization of the newly founded solution x^c is significantly lower than the current best solution (i.e. x^{*c}), then we update the current best solution as line 10. After that, the heuristic algorithm continues to find a new possible solution until the time limit has been reached. The parameter ϵ is defined to determine whether x^{*c} is updated and helps to reduce the number of rerouted flows. We can easily obtain the routing policy α_{ij}^k after getting the final solution x^{*c}.

5.4 Partial traffic prediction and compressive sensing

In the previous sections, we have presented the multi-time-steps segment routing (MTSR) strategy to reduce the number of rerouting flows while still achieving the TE target in a long time horizon. However, we need to cope with the following issues. First, the performance of MTSR relies on the accuracy of the traffic prediction model (i.e., GWN) which requires a large amount of data for the training and predicting processes. It leads to the high network monitoring cost problem which is often omitted in the prior ML-based TE studies. Second, MTSR imposes the GWN model to estimate the future traffic of all flows in the network, leading to the scalability problem when applying the method to a large-scale network. To overcome these problems, we propose an approach called MTSR-CS which combines the MTSR and the compressive sensing technique. In MTSR, we perform network measurement and traffic prediction only on a subset of flows. Then, the full matrix is reconstructed from the partial traffic prediction before being used to calculate the routing rules (by using the LS2SR algorithm).

5.4.1 Compressive sensing-based network traffic reconstruction

According to compressive sensing theory, the signal can be reconstructed or recovered from a few samples by exploiting the sparsity characteristic of the original signal. Therefore, compressive sensing can be used in reconstructing the network traffic from a few measurement data (Equation 5.31).

$$Z = \Phi X \tag{5.31}$$

where $X \in \mathbb{R}^{F\times 1}$ is a vector that contains the traffic volume of all flows. $F = N \times N$ is the total number of source-destination flows in the network (N is the number of nodes). Note that, instead of using a $N \times N$ matrix to represent the network traffic, it is represented by a vector that has F elements). $Z \in \mathbb{R}^{L\times 1}$ represents the measured traffic volumes. L is the number of flows that are measured ($L < F$). $\Phi \in \{0, 1\}^{L\times F}$ is a binary matrix to indicate the measured flows. Figure 5.3 shows an example of partial traffic measurement.

However, since network traffic X is not sparse in practice, compressive sensing cannot be directly applied to reconstruct X from Z. To overcome this problem, the authors in [22] use a transformation matrix $D \in \mathbb{R}^{F\times F}$ to sparsely project X in the transformation domain D (Equation 5.32).

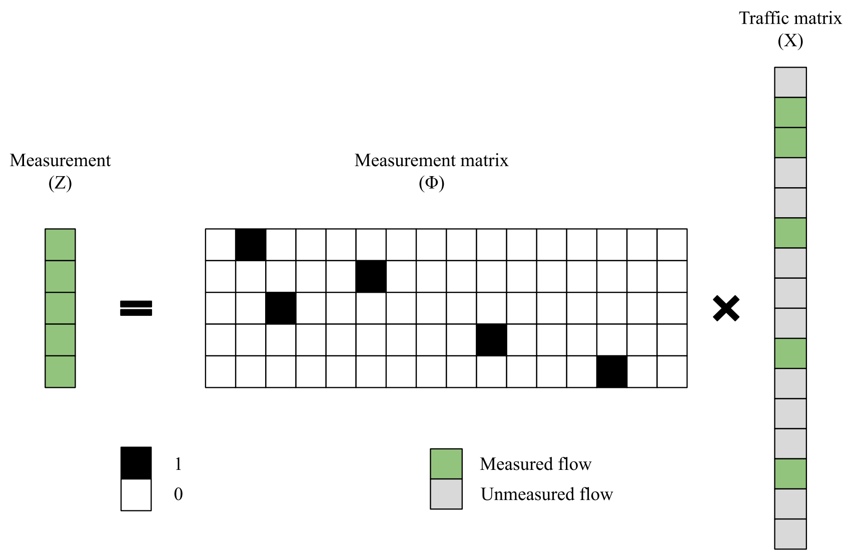


Figure 5.3: Partial traffic measurement. The traffic matrix X is represented as a vector.

$$X = DS \tag{5.32}$$

where $S \in \mathbb{R}^{F\times 1}$ is a sparse projection of X. S has K non-zero entries ($K < F$).

From Equation 5.31 and 5.32, we have:

$$Z = \Phi DS \tag{5.33}$$

Since S is a sparse vector, we can apply compressive sensing to reconstruct S from the measurement Z. Then the full network traffic X can be obtained by using Equation 5.32.

5.4.2 Reconstruct the traffic matrix from the partial traffic prediction

Unlike the study in [22], we do not use compressive sensing to reconstruct the network traffic itself but the maximum traffic demand from a partial traffic prediction. Figure 5.4 illustrates our idea on leveraging the proposed technique in [22] to reconstruct the predicted traffic matrix from the partial traffic prediction. The input of the routing algorithm (i.e., MTSR in Section 5.2) is a traffic matrix. The traffic matrix contains the predicted values of the maximum demands of all traffic flows in the next routing cycle (e.g., a routing cycle has T time-steps). Our idea to reduce the monitoring overhead is to reconstruct this matrix from a partial traffic prediction. First, we use a prediction model to estimate the maximum demands of L flows. Then, by applying compressive sensing, we can obtain the maximum traffic demands of all flows. Finally, we can calculate the routing rules using the LS2SR algorithm (i.e., Algorithm 2). Therefore, instead of monitoring and predicting the whole traffic matrix, we can monitor and predict the future demands of a subset of traffic flows and reduce the monitoring overhead.

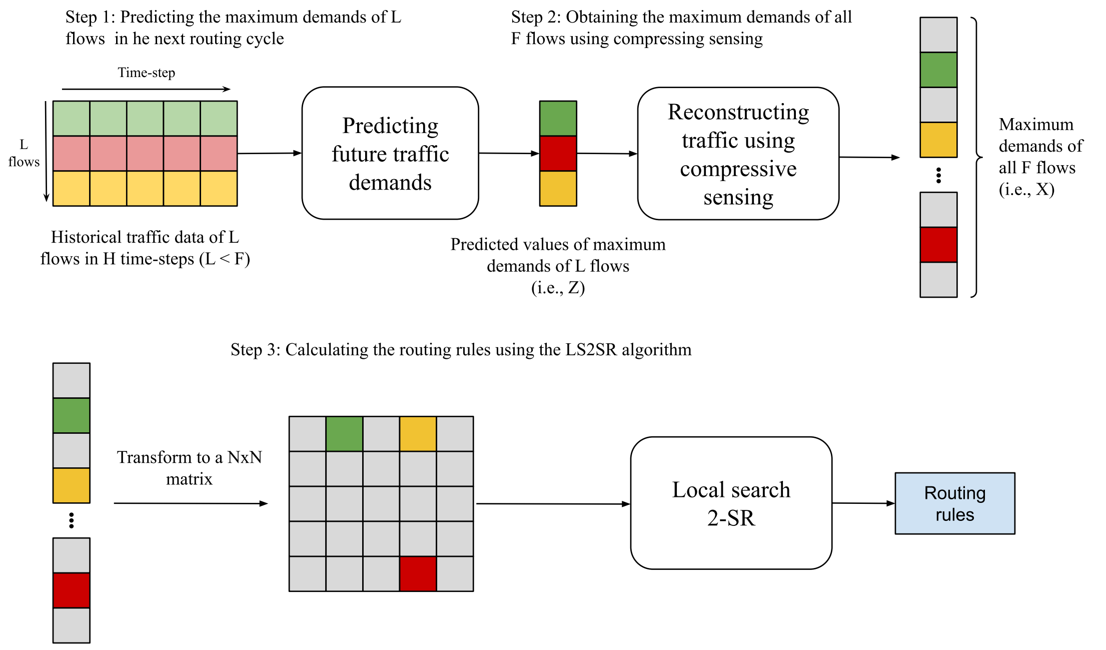

Step 1: Predicting the maximum demands of L flows in he next routing cycle
Time-step
L flows
Historical traffic data of L flows in H time-steps (L < F)
Predicting future traffic demands
Predicted values of maximum demands of L flows (i.e., Z)
Step 2: Obtaining the maximum demands of all F flows using compressing sensing
Reconstructing traffic using compressive sensing
Maximum demands of all F flows (i.e., X)
Step 3: Calculating the routing rules using the LS2SR algorithm
Transform to a NxN matrix
Local search 2-SR
Routing rules

Figure 5.4: Illustration of partial traffic prediction and matrix reconstruction using compressive sensing.

In Equation 5.31, Z is a vector whose elements are the predicted values of the maximum demands of L flows. X is the vector that contains the maximum demands of all F flows. The proposed method has two phases: the training phase and the testing phase. In the training phase, we use a training dataset to calculate the transformation matrix D and train the prediction model. In the testing phase, at the beginning of every routing cycle, we perform partial traffic prediction of the subset of flows. The maximum demands of other flows will be reconstructed using compressive sensing. Then, the routing rules are calculated using the LS2SR algorithm. The details will be described in the following sections.

5.4.3 Training phase

There are two tasks in the training phase: obtaining transformation matrix D and training the prediction model. The training dataset contains the historical measured traffic volume of all flows. However, since our interest is the maximum demand in a routing cycle, the original dataset will be transformed into the set of maximum traffic matrices as shown in Figure 5.5. We generate the maximum matrix by taking the maximum values of each flow in every routing cycle (T time-steps). The matrix is flattened to a vector that has F elements. Then, we use the generated data to learn the transformation matrix D and train the prediction model.

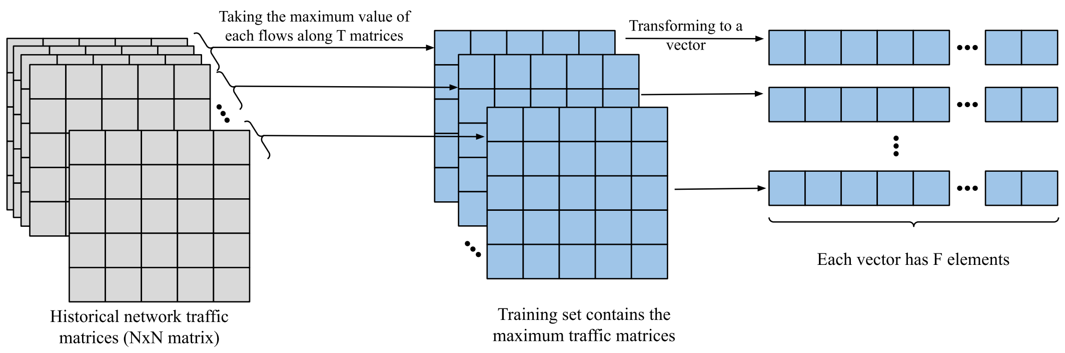

Taking the maximum value of each flows along T matrices
Transforming to a vector
Historical network traffic matrices (NxN matrix)
Training set contains the maximum traffic matrices
Each vector has F elements

Figure 5.5: Generating the training dataset from network traffic matrices.

We can get the transformation matrix D by solving the following optimization problem:

$$\text{minimize} \quad ||\hat{X} - D\hat{S}||^2 \tag{5.34}$$

$$||\hat{S}(i)||_0 \quad \leq \quad K \tag{5.35}$$

$$\hat{S}(i) \quad \geq \quad 0 \tag{5.36}$$

$$i = 1, 2, .., T \tag{5.37}$$

where $D \in \mathbb{R}^{F\times F}$ and $\hat{S} = (\hat{S}(1), \hat{S}(2), ..., \hat{S}(T)) \in \mathbb{R}^{F\times T}$ are two unknown variables. $\hat{X} = (\hat{X}(1), \hat{X}(2), ..., \hat{X}(T)) \in \mathbb{R}^{F\times T}$ represents T vectors in the training set. Each column $\hat{S}(i) \subset \hat{S}$ is the sparse representation of the traffic vector $\hat{X}(i)$. K is the upper limit of the number of non-zero entries in the sparse representation.

According to [22], we can use the Alternative Least Square (ALS) to obtain the solution to the above problem. First, we randomly initialize values for the transformation matrix D. Then, given D, we use ALS to determine $\hat{S}$. Next, based on the obtained $\hat{S}$, we can update the values of D. $\hat{S}$ and D are iteratively updated until reaching the terminated conditions (e.g., the maximum iteration). The updating process of D is described as follows. We will update one column of the transformation matrix D at a time. Let d_k be the k^{th} column of D and $\hat{s}_k$ be the k^{th} row of $\hat{S}$ (where $k = 1, 2, ..., F$). The multiplication $D\hat{S}$ is divided into the sum of M rank-1 matrices: $D\hat{S} = \sum_{j=1}^{M} d_j\hat{s}_j$. We have:

$$\hat{X} - D\hat{S} = \hat{X} - \sum_{j=1}^{M} d_j\hat{s}_j = (\hat{X} - \sum_{j\neq k}^{M} d_j\hat{s}_j) - d_k\hat{s}_k = E_k - d_k\hat{s}_k \tag{5.38}$$

Then, to update the k^{th} column of D, we solve the following optimization problem:

$$\begin{aligned} &\text{minimize} \quad ||E_k - d_k\hat{s}_k||^2 && (5.39)\\ &||\hat{S}(i)||_0 \quad \leq \quad K && (5.40)\\ &\hat{s}_k \quad \geq \quad 0 && (5.41)\\ &i = 1, 2, .., T; k = 1, 2, ..., F && (5.42) \end{aligned}$$

We use Singular Value Decomposition (SVD) to update d_k and $\hat{s}_k$. For the details of solving the above problem, please refer to [22]. Then, we repeat the process above to update other columns of D.

5.4.4 Testing phase

There are three tasks in the testing phase: predicting maximum future traffic of L flows (i.e., Z), reconstructing vector X, and calculating network routing rules. The network controller will perform these tasks at the beginning of every routing cycle. First, we need to decide the subset of flows to be monitored and predicted. The authors in [69] show that among the flows in the network, there are so-called “critical flows" that may have a huge impact on the performance of TE when rerouting them. Therefore, we choose the top $k\%$ of the largest flows based on their historical volumes ($k = L/F$). This method considerably reduces the monitoring cost since only the top $k\%$ of flows (top-k) are monitored rather than all flows. Note that the top-k largest flows in the training data and testing data may be different.

After that, we adapt the GWN model in [59] to directly predict the maximum demand of each flow in the top-k. Hence, only top-k flows need to be trained and predicted, which reduces the computational burden and increases the scalability of the prediction model.

Then, we reconstruct the vector X from the partial prediction. Let $Z \in \mathbb{R}^{L\times 1}$ be the predicted traffic values. We solve the following problem to find the sparse representation S using the transformation matrix D, the measurement matrix Φ, and the predicted traffic vector Z.

$$S = \text{argmin}||S||_0 \tag{5.43}$$

$$Z = \Phi D S \tag{5.44}$$

$$S \geq 0 \tag{5.45}$$

Then, we get the reconstruction results of X using Equation 5.32.

Dataset	No. nodes	No. links	Monitoring granularity
Abilene	12	30	5 (min)
Geant	22	72	15 (min)

Table 5.2: Details of the real traffic datasets.

5.5 Performance evaluation

5.5.1 Datasets and performance metrics

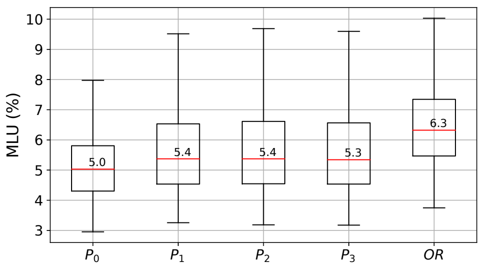
We conduct experiments on two real datasets: Abilene and Geant, available at [36]. The details of the datasets are shown in Table 5.2. We use one month of data for training the prediction model and two weeks of data for testing. To evaluate the performance of our proposed methods in cases of large-scale networks, we use REPETITA dataset [15]. The dataset contains more than 200 topologies of the real backbone network, whose number of nodes varies from 4 to more than 100. For each topology, five traffic matrices were generated and adjusted so that the optimal values of the MLU (obtained from solving the MCF problem) are around 90%. This is the same dataset that was used to evaluate the scalability of the SRLS approach in [14].

We evaluate the performance of our proposed approach using two main metrics: the maximum link utilization (MLU), i.e., Equation 5.46, and the average number of routing changes (RC) per time-step. After obtaining the routing policy, MLU is calculated using the actual traffic matrix from the test set. RC is the total number of flows that are rerouted after each routing cycle.

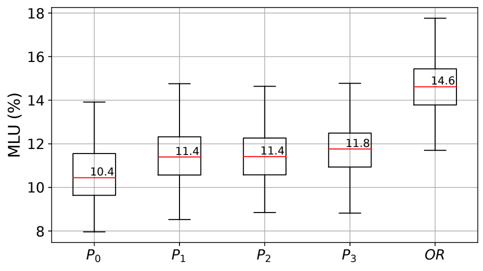
$$MLU = \max_{e \in E} \frac{load(e)}{capacity(e)} \tag{5.46}$$

5.5.2 Evaluating the MTSR approach without traffic prediction

In the first experiment, we evaluate the performance of the routing policy acquired from the five TE problems P_0, P_1, P_2, P_3, and Traffic Matrix Oblivious Segment Routing (OR) [5]. Note that we solve the problem P_0 for every time-step and only solve the OR problem once. The objective of this experiment is to evaluate the performance bound of the three MTSR approaches (i.e., P_1, P_2, and P_3) in the best scenarios by removing the impacts of the prediction model. In this experiment, we assume that the future

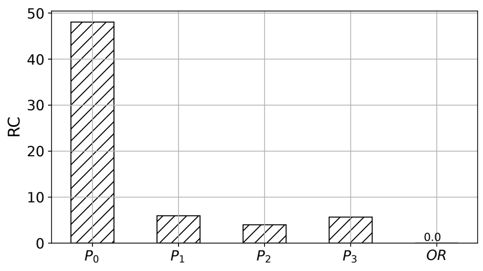


(a) MLU of 100 time-steps - Abilene network.

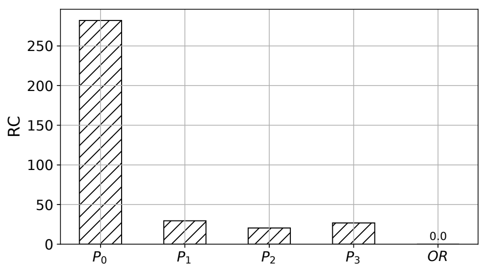


(b) MLU of 100 time-steps - Geant network.

Figure 5.6: The maximum link utilization with different routing approaches.



(a) Abilene network.



(b) Geant network.

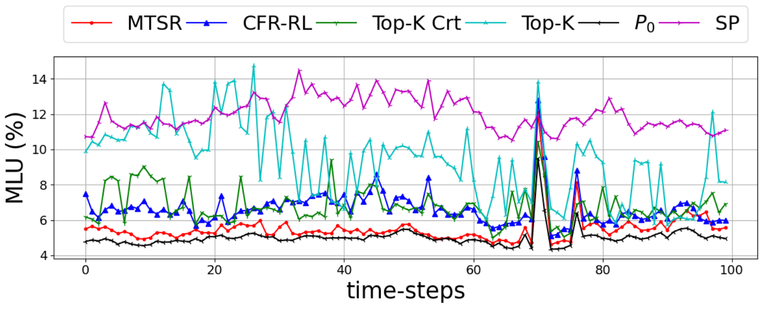
Figure 5.7: The number of routing changes of different routing approaches.

traffic matrices can be accurately predicted. Therefore, we use the real traffic matrices from the test set as the inputs for solving the TE problem. We use the solver from the PuLP library [50] to solve all the problems and obtain the optimal solutions. In all experiments, the routing cycle length T for MTSR schemes is set to 12.

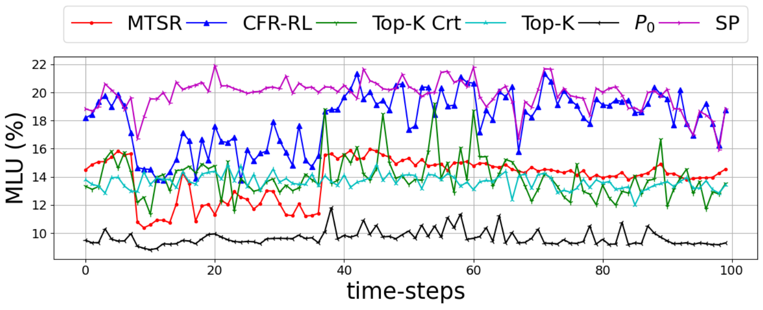
Figure 5.7 shows the results of all approaches on the Abilene and Geant datasets. Obviously, by solving the TE problem at every time-step, P_0 achieves the best results in terms of MLU on all datasets. However, although P_0 has a slightly better MLU compared to P_1, P_2, and P_3 (e.g., about 8% on the Abilene network), it suffers from a significantly high number of routing changes. The Oblivious Routing (OR) shows the largest value of MLU on both datasets. Especially on the Geant network, the average MLU obtained by OR is about 20% higher than that of other approaches.

As mentioned in the theoretical analysis (Section 5.2.2), among the three formulations of the MTSR problem, P_1 has the best performance in comparing the MLU, P_3 and P_2 have come the second and third, respectively. However, the gaps among them are relatively small. Although all of the MTSR approaches have a similar number of routing changes, P_2 reveals the smallest number of routing changes in both datasets. Based on the observations, we can see four merits for solving the MTSR problem with P_2 formulation: (a) reducing the problem complexity, (b) alleviating the difficulty in the traffic prediction task, (c) reducing the number of rerouting, and (d) achieving comparable performance with other approaches. Therefore, in the rest of this chapter, the presented results of the MTSR are obtained by solving the problem P_2.

5.5.3 Evaluating the MTSR approach with traffic prediction



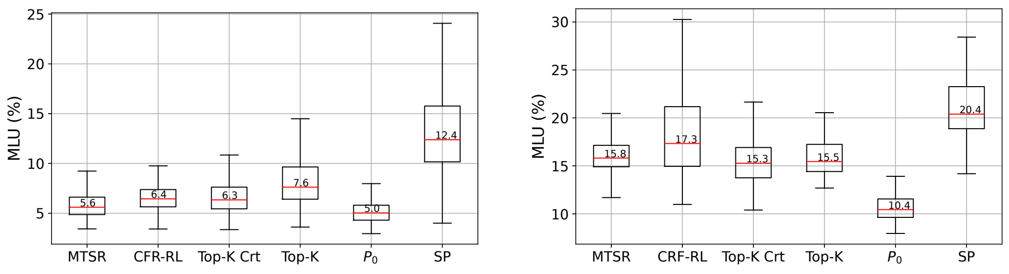
(a) MLU of 100 time-steps - Abilene network.



(b) MLU of 100 time-steps - Geant network.

Figure 5.8: The maximum link utilization of the first 100 time-steps.

In this experiment, we evaluate the performance of the proposed MTSR approach (P_2) as an online segment routing algorithm using predicted traffic. We assume that all traffic flows can be monitored. We train a state-of-the-art deep learning model called Graph WaveNet (GWN) [59] and use the trained model for predicting future traffic matrices. The prediction model uses historical data of H time-steps to predict the matrices of maximum demand of each flow in the next routing cycle. The training details are omitted but can be found at [2]. At the beginning of each cycle, after obtaining the predicted traffic matrices, we solve the problem and acquire the routing policies. Then, the actual traffic matrix is used to calculate the maximum link utilization for every time-step.



(a) Abilene network.

(b) Geant network.

Figure 5.9: The maximum link utilization of different routing approaches.

We use the proposed LS2SR (Algorithm 2 in Section 5.3) for solving the MTSR problem, given the predicted traffic as the input. We set the length of the routing cycle as $T = 12$ and the number of historical steps used for the prediction model as $H = 2T$. We set $\beta = 16$ and $\gamma = 1$, which are adopted from [14].

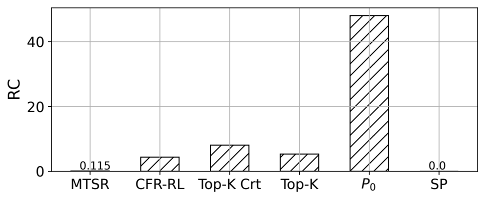
We compare the performance of our proposed MTSR scheme with five other baselines:

- **CFR-RL**: in each time-step, a set of critical flows (top $k\%$) is selected among the traffic matrix using a Deep Reinforcement Learning algorithm. The selected flows are rerouted by solving the TE problem while the paths of remained flows are unchanged.

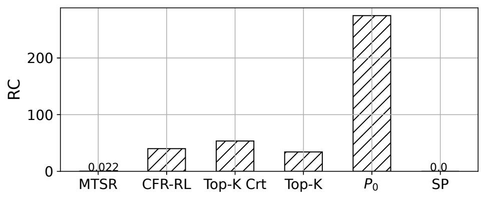
- **Top-K**: in each time-step, select top $k\%$ largest flows from the given traffic matrix. Similar to the CFR-RL method, only selected flows are rerouted based on the solution obtained from solving the TE problem.

- **Top-K Critical (Top-K Crt)**: similar to the Top-K approach, but it selects the top $k\%$ largest flows from the five most congested links.

- ***P_0***: the TE problem is solved in every time step by using the solver from the PuLP library [50] (same as in Experiment 1).
- **Shortest Path (SP)**: all traffic flows are routed using the shortest paths.



(a) Abilene network.



(b) Geant network.

Figure 5.10: The average number of rerouting flows per time-step.

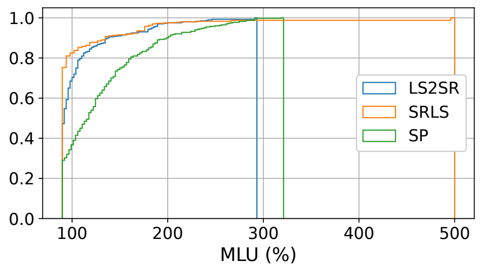
The CFR-RL, Top-K, and Top-K Critical are adopted from [69]. Based on the results in [69], the value of $k\%$ equals to 10%. Note that, in [69], new routing paths of the critical flows are obtained by solving the MCF problem. To have a fair comparison, we solve it using 2-segment routing. Moreover, to evaluate the performance of the routing algorithm under the same time constraint, the solving time of all the routing methods is limited to 10s. The experiment results are depicted in figures 5.9 and 5.10.

In summary, by applying MTSR, we significantly reduce the number of traffic flows need to be rerouted per time-step while still achieving an acceptable performance concerning the MLU.

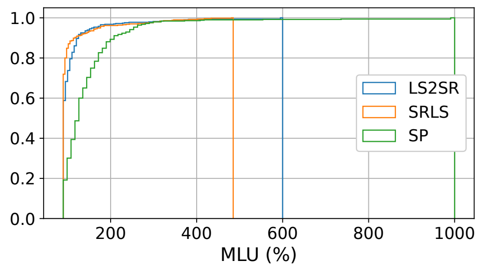
Figures 5.8(a) and 5.8(b) illustrate the MLU in the first 100 time-steps while Figures 5.9(a) and 5.9(b) show the overall results of all methods. With the Abilene dataset, our method attains better performance than that of other approaches (excepting P_0). In contrast, Top-K achieves the lowest MLU on the Geant network. Both MTSR and CFR-RL show a degradation in the performance due to the significant increase in the number of flows in the network (from 144 to 484). The ML-based approaches reveal limitations when dealing with large-scale networks.

Figure 5.10, on the other hand, shows similar results for the two networks in term of the number of rerouting flows per time-step. In MTSR, the flow paths keep unchanged within T time-steps of the routing cycle. CFR-RL, Top-K, and Top-K Critical result in 10 and 40 rerouting flows per time-steps in the two networks since they perform the rerouting of at most 10% flows at every time-step.

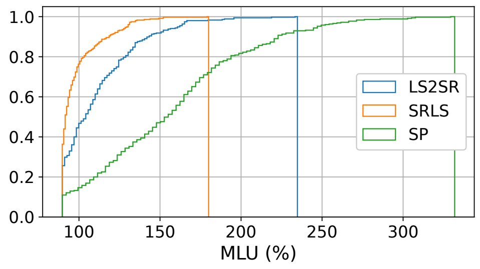
5.5.4 Evaluating the scalability of the LS2SR algorithm



(a) Small networks (nodes < 20).



(b) Medium networks (20 ≤ nodes < 40).

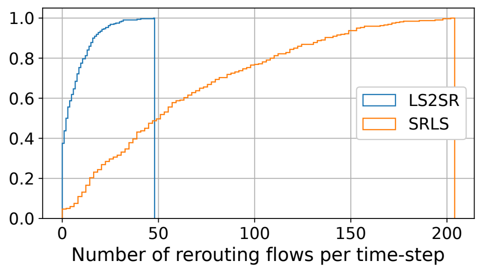


(c) Large networks (40 ≤ nodes).

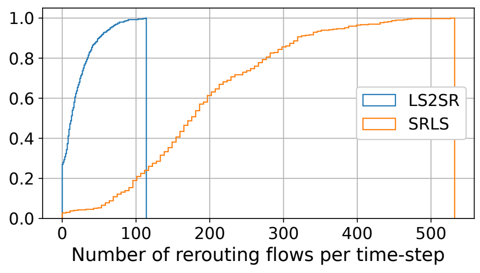
Figure 5.11: The maximum link utilization of different groups of network topology.

In order to evaluate the scalability of our LS2SR algorithm (Algorithm 2 proposed in Section 5.3), we test the performance using different sized networks from REPETITA dataset [15] in this experiment. Similar to the experiment in [14], we divide the dataset into three groups based on the number of nodes in the network topology. Group 1 comprises networks that have less than 20 nodes. The number of nodes in Group 2 varies from 20 to 40. Group 3 contains large networks which have more than 40 nodes. We compare our results with SRLS in [14] and Shortest Path (SP) routing approach.

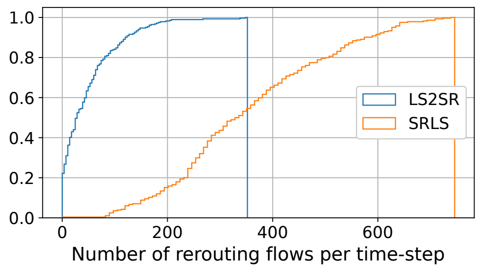
Note that SRLS uses n-segment routing for solving the TE problem while LS2SR only uses 2-segment routing. We conduct the experiment using five synthetic traffic matrices of each topology and calculate the MLU, the number of rerouting flows per time-step (RC) and the average delay as performance metrics. The results are showed in Figures 5.11, 5.12, and 5.13 respectively. The delay is computed by averaging the delay of all the traffic flows. The delay of a traffic flow is calculated as the sum of link delays in its path. The link delays provided in the REPETITA dataset represent the



(a) Small networks (nodes < 20).



(b) Medium networks (20 ≤ nodes < 40).



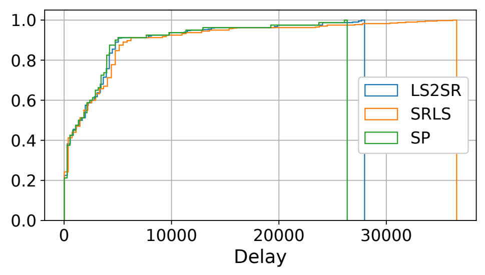
(c) Large networks (40 ≤ nodes).

Figure 5.12: The number of rerouting flows per time-step of different groups of network topology.

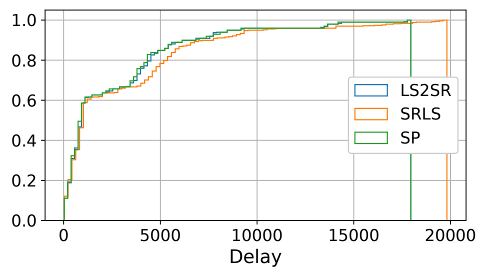
geometrical distance between two nodes. Note that since SP has zero rerouting flow, its results are not showed in Figure 5.12.

Our proposed approach LS2SR achieves the same performance in term of MLU compared with SRLS. However, LS2SR has the best results in reducing the number of rerouting flows, and has similar results compared with SP, and is consistently better than SRLS concerning the average delay metric.

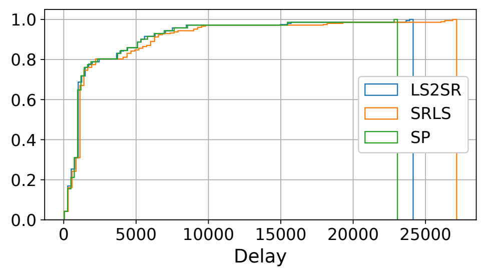
In our experiments, the maximum link utilization is calculated by using Equation 5.46. The traffic load on link e is computed by taking the sum of the traffic volume of all the flows that pass through link e. Since we do not run the network simulation, depending on the routing rules, the traffic load can be larger than the link capacity. Therefore, in some cases, the MLU can be larger than 1 or 100%. Especially in the experiment with the REPETITA dataset, the traffic matrices are scaled so that the maximally utilized link is loaded at 90% of its capacity in the optimal solution of the corresponding multi-commodity flow problem.



(a) Small networks (nodes < 20).



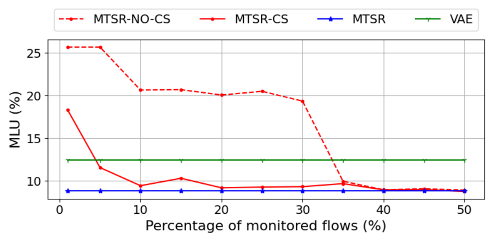
(b) Medium networks (20 ≤ nodes < 40).



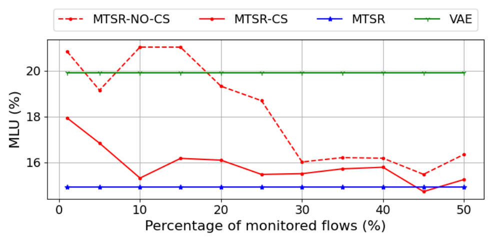
(c) Large networks (40 ≤ nodes).

Figure 5.13: The average delay per time-step of different groups of network topology.

5.5.5 Evaluating the MTSR approach with compressive sensing



(a) MLU on Abilene dataset.



(b) MLU on Geant dataset.

Figure 5.14: The MLU of MTSR-CS with different percentages of monitored flows.

This experiment is designed to evaluate the performance of the approach combining MTSR for TE and Compressive Sensing for traffic matrix construction. The number of monitored flows varied from 1% to 50% of all the flows in the network. We compare our proposed approach, namely MTSR-CS, with the following methods:

- **MTSR-NO-CS**: We apply the MTSR scheme to the predicted values of the critical flows without using CS to reconstruct the full traffic matrix. The missing values are replaced by zero.
- **VAE** [23]: Traffic matrix is reconstructed from fully monitored link loads with the support of the VAEs model.
- **MTSR**: We use the same method in Experiment 2, where all the traffic flows are monitored and predicted.

Figures 5.14(a) and 5.14(b) illustrate the performance comparison of all methods concerning MLU metric. The figures show that with a medium-sized network (e.g., Abilene network), MTSR-CS can achieve a similar performance as MTSR while it only measures 10% of total flows. Considering the case of the Geant dataset, it requires more than 40% of monitoring flows to achieve the same performance as MTSR. While MTSR-CS continues to outperform MTSR-NO-CS, the performance gap between them has greatly declined as the percentage of monitored flows increases. VAE method achieves relatively good results in the small network (i.e, Abilene network) while degrading its performance in the larger network (i.e., Geant network). Overall, by using compressive sensing to reconstruct the traffic matrix from the partially monitored data, we can reduce the monitoring overhead while maintaining the routing performance.

5.6 Summary

In this chapter, we studied the multi-time-step segment routing (MTSR) with traffic prediction and addressed two problems: a large number of traffic rerouting and high traffic monitoring overhead. We leveraged the traffic prediction for performing traffic engineering while minimizing the number of flows that need to be rerouted. By considering the problem complexity and the difficulty of multi-step traffic prediction, we formulated three versions of the MTSR problem. We also provided a theoretical analysis of the solution of these three formulations. Moreover, we proposed an efficient algorithm for quickly solving the multi-step segment routing and further reducing the number of routing changes. To reduce the monitoring cost, only a set of critical flows need to be monitored and predicted. And the compressive sensing technique was used to reconstruct the full matrix from the partially predicted values. Our evaluation on real network datasets showed that our proposed approach can significantly reduce the number of routing changes, and meet the requirements for the maximum link utilization. Besides that, the experimental results of MTSR-CS show that we can reduce at least 50% of monitoring cost while still achieving the same performance as the MTSR approach.

6 Conclusion

6.1 Summary of the dissertation

Traffic engineering (TE) is one of the most critical problems throughout the development of the Internet, which takes responsibility to facilitate efficient and reliable network operations while simultaneously optimizing network resource utilization and traffic performance. With the rapid growth in the amount of network traffic as well as the dynamic traffic behavior, artificial intelligence and machine learning techniques are believed to be the key technologies to overcome the limitations of traditional solutions in TE. However, we face many challenges when applying ML/DNN to networking problems. In this dissertation, we have addressed three problems in the traffic matrix prediction and traffic engineering problems leveraging machine learning techniques.

In the first part of this dissertation, we propose a new method, named FWBW-LSTM, for estimating the future traffic demands when the missing values occur in the monitored traffic data. Due to the accumulative error caused by missing values, the existing prediction models result in low prediction accuracy. To overcome this problem, based on the Bidirectional LSTM network, we propose the data correction algorithm to correct the imprecise values in the predicted outputs. In addition, we introduce the method for selecting the traffic flows that need to be monitored. With the proposed approach, we can outperform other well-known methods for time series prediction.

In the second part, we propose GCRINT, a graph-based deep neural network, to impute the missing values in the historical network dataset. From the first part, we observe that training the deep learning model with a dataset that contains missing values will degrade its performance. Our approach leverages the graph neural network to learn the dynamic, spatial relations among the traffic flows. Along with the temporal relations which are extracted by using the LSTM network, the spatial relation helps to increase the accuracy of the imputed values. Our experiment's results on traffic engineering show that our method can help to improve the performance of network routing over state-of-the-art approaches.

In the remaining, we address the problem of high frequent traffic rerouting in traditional TE solutions. We proposed a traffic prediction-aided traffic engineering approach, named MTSR. We utilize the multi-time-step traffic prediction and formulate the routing problem by considering the worst-case scenario. The TE is only performed after a routing cycle (which includes multiple time-steps) to reduce the number of traffic flows rerouting over the long run. Our proposed approach can accomplish two objectives: reducing the number of traffic flows rerouting and achieving an acceptable maximum link utilization which is an important metric in traffic routing. In addition, we also consider the scenario when only a few traffic flows are measured (to reduce monitoring cost). We apply compressive sensing to reconstruct the monitored data before applying the MTSR method. The approach, which is named MTSR-CS, can reach the same performance as MTSR while reducing at least 50% monitoring cost.

6.2 Discussion and open issues

6.2.1 Network traffic datasets

In this research, we focused on developing DNN models which are used for estimating the missing values and predicting the future demands in network traffic. In most of our proposed methods, the network dataset is an important component and has a direct impact on the performance and applicability of the DNN models. The performance evaluations so far were conducted by using well-known historical traffic datasets such as Abilene and Geant network datasets [36]. However, there is no standardized dataset for network traffic prediction problems. Besides, due to privacy and lack of shareable data, there are a few available datasets that can be used in traffic prediction. Most of the datasets are old and insufficient to train DNN models. Therefore, the trained models may not be efficiently applied to the complexity increases and behavior of the current modern networks (e.g., 5G/6G systems).

The synthetic datasets which are generated by mathematical model [53] or network simulation tools [44, 55] can be an alternative source of network datasets. Besides, by using synthetic datasets, the experiments can be run with different network scales to evaluate the scalability of the proposed methods. However, data-driven-based methods like ML/DNN may not be benefited from synthetic data. Generating network traffic from a mathematical model seems to follow a probabilistic distribution and may not reflect the dynamic network behavior. For example, the most common assumption is that the source-destination flows are uniformly distributed [34]. Therefore, while synthetic datasets can be used in the performance comparison among ML/DNN-based methods, the trained models may not be efficiently applied in real network scenarios.

6.2.2 Explainable ML/DNN techniques

In this study, we leverage DNN models as '*black box*' algorithms to overcome the limitations of existing solutions. Although the experiment's results show that our proposed models can outperform the existing solutions (e.g., improving prediction accuracy), the machine learning-based approaches suffer several limitations. For example, being used as '*black box*' algorithm, machine learning has been criticized for its limits in extracting knowledge from data, especially in obtaining a performance guarantee. Since machine learning models are constructed and tested using a specific dataset, the performance may not be replicated in others. In addition, parameters tunning in the training phase can have a huge impact on the overall performance of the DNN model (e.g., the size of data samples/features, and the number of DNN layers). Therefore, understanding and/or explaining ML techniques to provide a stronger descriptive approach can be the potential direction. This research directly relates to an open topic in the machine learning area called ‘*’Explainable AI*’. Although many scientists focus on interpreting the behavior of the ML model, ‘*’Explainable AI*’ remains a huge challenge, especially in the context of modern deep learning. The explainability of a machine learning model is usually inverse to its prediction accuracy - the higher the prediction accuracy, the lower the model explainability [66].

6.2.3 Traffic prediction accuracy versus network routing performance

Our objective in this study is to address the practical problem when applying machine learning models for network traffic prediction and network routing. The experiment's results show that higher prediction accuracy usually leads to higher performance in network routing. We focus on improving the prediction accuracy of the ML model without considering the relations between the outputs of traffic prediction and the routing algorithm. In some cases such as low traffic load, the network routing algorithm does not always require high traffic prediction to achieve acceptable routing performance. Therefore, in our study (Section 5), instead of trying to accurately forecast the traffic demands at every time-step, we predict the maximum traffic volume in the near future. The experiment's results show that by preparing the worst-case scenario (i.e., maximum traffic demands), we can achieve similar performance as accurately knowing the exact traffic demands in the future.

However, it is helpful to study the relationship between prediction accuracy and the performance of network routing. By investigating this problem, we can estimate the outcome of the routing problem under the circumstance of increasing or decreasing the prediction accuracy. In addition, we can develop a new routing approach that takes into consideration the possibility that the machine learning prediction can be inaccurate. For example, in [25], the authors devise algorithms for traffic engineering in IP networks with (possibly) inaccurate traffic matrix prediction.

6.3 Future direction

In this dissertation, we have addressed the TE problem in the backbone network where the proposed approaches are expected to be executed in a centralized network controller (e.g., SDN controller). Although the centralized approaches can benefit from the global information, these methods show limitations in high monitoring/communicating costs (e.g., collecting traffic data, exchanging information among network devices and the controller). These problems become crucial in the 6G network where we have an extremely large-scaled network (hundreds of network nodes in the core layer and billions of connected mobile devices at the edges network). In addition, to cope with the increase in the size of both physical network and traffic demands, it required more complicated ML models to learn the network behavior.

Recently, a class of machine learning techniques called Deep Reinforcement Learning (DRL) has showed great potential in solving network control problems. With the abilities of self-learning and online exploration, DRL stands out among the AI/ML techniques as a promising method to address the problem of a highly complex and dynamic network environment. In DRL, an agent will iteratively interact with the network environment to learn a policy that can maximize a designed reward. In traffic engineering, Deep Reinforcement Learning is a promising technique to construct a self-adaptive network routing mechanism [52, 67]. Furthermore, the DRL-based routing mechanism can be extended to work in the distributed approach by using the Multi-Agent Reinforcement Learning technique (MARL). MARL includes many agents which need to learn to cooperate for maximizing a common reward. In the network routing problem, each agent can be considered as a network node and can independently decide the path for the incoming traffic [17, 39, 51]. Since each agent may only need to take responsibility for solving a small part of the routing problem, MARL can reduce the model complexity, and deduct the communication overhead compared to the centralized approach.

However, applying MARL in traffic routing is still in an early stage. Most of the proposed solutions focus on hop-by-hop routing while each network device is considered an independent agent in the MARL system. The agent will decide the next destination for the incoming traffic without any cooperation among agents [17, 39]. To provide the self-adaptive and cooperative network routing mechanism, the following problem can be considered when applying MARL to traffic routing:

1. **Traffic routing in large-scale networks**: Although considering a network node (e.g., router) as an agent can be considered a natural approach in network

routing, training a large number of agents in a network with thousands of nodes requires high computational resources. In addition, the overall performance of the system may degrade due to the difficulty in cooperation between the agents. To address this problem, hierarchical routing using MARL techniques can be considered. In hierarchical routing, each agent can take control in a small region of the network. The high-level agents will have responsibility for forwarding the traffic between the regions.

2. **Cooperation problem in MARL-based traffic routing**: Cooperation is one of the most challenging problems when directly applying MARL to network routing. In a naive solution, each router or network node can become an agent and decide the paths for the incoming traffic based on its information. The agent treats other agents as part of the network and does not aware of their policies and actions. Since agents are not trained to cooperate, their actions or routing rules may have conflicts and create problems in network operation. To overcome this, the centralized training and decentralized execution approach (CTDE) [42] can be used in training the MARL. Therefore, the agent can learn how to take actions (i.e., generate routing rules) that can maximize the global reward such as maximizing the overall traffic throughput or minimizing traffic congestion.

This research direction is expected to address the challenges in the end-to-end network routing problems of the large-scale and dynamic network environment. Therefore, an automatic, adaptive routing mechanism can be developed to provide end-to-end QoS in beyond 5G and 6G systems. The results may open new potential research directions on integrating AI/ML techniques to satisfy the requirements of the 6G network and to boost the progress of applying ML in the networking field in general.

Bibliography

[1] https://github.com/vananle/GCRINT.

[2] https://github.com/vananle/MTSR.

[3] Yossi Azar et al. "Optimal Oblivious Routing in Polynomial Time". In: *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*. STOC '03. San Diego, CA, USA: Association for Computing Machinery, 2003, pp. 383–388. ISBN: 1581136749. DOI: 10.1145/780542.780599. URL: https://doi.org/10.1145/780542.780599.

[4] Abdelhadi Azzouni and Guy Pujolle. "NeuTM: A neural network-based framework for traffic matrix prediction in SDN". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. 2018, pp. 1–5. DOI: 10.1109/NOMS.2018.8406199.

[5] Randeep Bhatia et al. "Optimized network traffic engineering using segment routing". In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. 2015, pp. 657–665. DOI: 10.1109/INFOCOM.2015.7218434.

[6] George EP Box et al. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[7] Wei Cao et al. "Brits: Bidirectional recurrent imputation for time series". In: *Advances in Neural Information Processing Systems*. 2018, pp. 6775–6785.

[8] Xiaofeng Cao et al. "Interactive Temporal Recurrent Convolution Network for Traffic Prediction in Data Centers". In: *IEEE Access* 6 (2018), pp. 5276–5289. DOI: 10.1109/ACCESS.2017.2787696.

[9] Radu Cârpa et al. “Evaluating the impact of SDN-induced frequent route changes on TCP flows”. In: *2017 13th International Conference on Network and Service Management (CNSM)*. 2017, pp. 1–9. DOI: 10.23919/CNSM.2017.8256021.

[10] Zhengping Che et al. “Recurrent neural networks for multivariate time series with missing values”. In: *Scientific reports* 8.1 (2018), pp. 1–12.

[11] Anja Feldmann et al. “The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic”. In: *Proceedings of the ACM Internet Measurement Conference*. IMC ’20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1–18. ISBN: 9781450381383. DOI: 10.1145/3419394.3423658. URL: https://doi.org/10.1145/3419394.3423658.

[12] Open Networking Fundation. “Software-defined networking: The new norm for networks”. In: *ONF White Paper* 2.2-6 (2012), p. 11.

[13] Zhen Gao et al. “Compressive Sensing Techniques for Next-Generation Wireless Communications”. In: *IEEE Wireless Communications* 25.3 (2018), pp. 144–153. DOI: 10.1109/MWC.2017.1700147.

[14] Steven Gay, Renaud Hartert, and Stefano Vissicchio. “Expect the unexpected: Sub-second optimization for segment routing”. In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 2017, pp. 1–9. DOI: 10.1109/INFOCOM.2017.8056971.

[15] Steven Gay, Pierre Schaus, and Stefano Vissicchio. “Repetita: Repeatable experiments for performance evaluation of traffic-engineering algorithms”. In: *arXiv preprint arXiv:1710.08665* (2017).

[16] Arpit Gupta et al. “Sonata: Query-driven streaming network telemetry”. In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM. 2018, pp. 357–371.

[17] Bo He et al. “Deephop on edge: Hop-by-hop routing bydistributed learning with semantic attention”. In: *49th International Conference on Parallel Processing-ICPP*. 2020, pp. 1–11.

[18] David Hong, Tamara G Kolda, and Jed A Duersch. “Generalized canonical polyadic tensor decomposition”. In: *SIAM Review* 62.1 (2020), pp. 133–163.

[19] Jonghwan Hyun, Nguyen Van Tu, and James Won-Ki Hong. "Towards knowledge-defined networking using in-band network telemetry". In: *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2018, pp. 1–7.

[20] Gianluca Iannaccone et al. "Analysis of Link Failures in an IP Backbone". In: *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*. IMW '02. Marseille, France: Association for Computing Machinery, 2002, pp. 237–242. ISBN: 158113603X. DOI: 10.1145/637201.637238. URL: https://doi.org/10.1145/637201.637238.

[21] Mathieu Jadin et al. "CG4SR: Near Optimal Traffic Engineering for Segment Routing with Column Generation". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 2019, pp. 1333–1341. DOI: 10.1109/INFOCOM.2019.8737424.

[22] Dingde Jiang et al. "A Compressive Sensing-Based Approach to End-to-End Network Traffic Reconstruction". In: *IEEE Transactions on Network Science and Engineering* 7.1 (2020), pp. 507–519. DOI: 10.1109/TNSE.2018.2877597.

[23] Grigorios Kakkavas et al. "Future Network Traffic Matrix Synthesis and Estimation Based on Deep Generative Models". In: *2021 International Conference on Computer Communications and Networks (ICCCN)*. 2021, pp. 1–8. DOI: 10.1109/ICCCN52240.2021.9522222.

[24] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations (ICLR)*. 2017.

[25] Murali Kodialam and T. V. Lakshman. "Prediction Augmented Segment Routing". In: *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. 2021, pp. 1–6. DOI: 10.1109/HPSR52026.2021.9481858.

[26] V. A. Le, P. Le Nguyen, and Y. Ji. "Deep Convolutional LSTM Network-based Traffic Matrix Prediction with Partial Information". In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2019, pp. 261–269.

[27] Van An Le et al. "GCRINT: Network Traffic Imputation Using Graph Convolutional Recurrent Neural Network". In: *ICC 2021 - IEEE International Conference on Communications*. 2021, pp. 1–6. DOI: 10.1109/ICC42927.2021.9500687.

[28] Ming-Chieh Lee and Jang-Ping Sheu. “An efficient routing algorithm based on segment routing in software-defined networking”. In: *Computer Networks* 103 (2016), pp. 44–55.

[29] Yaguang Li et al. “Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting”. In: *International Conference on Learning Representations (ICLR ’18)*. 2018.

[30] Yonghong Luo et al. “E 2 GAN: end-to-end generative adversarial network for multivariate time series imputation”. In: *28th International Joint Conference on Artificial Intelligence*. 2019, pp. 3094–3100.

[31] N. McKeown et al. “OpenFlow: enabling innovation in campus networks”. In: *ACM SIGCOMM Computer Communication Reviemckeown2008openfloww* 38.2 (2008), pp. 69–74.

[32] Jim McManus et al. *Requirements for Traffic Engineering Over MPLS*. RFC 2702. Sept. 1999. DOI: 10.17487/RFC2702. URL: https://rfc-editor.org/rfc/rfc2702.txt.

[33] L. Nie et al. “Traffic matrix prediction and estimation based on deep learning in large-scale IP backbone networks”. In: *Journal of Network and Computer Applications* 76 (2016), pp. 16–22.

[34] Antonio Nucci, Ashwin Sridharan, and Nina Taft. “The Problem of Synthetically Generating IP Traffic Matrices: Initial Recommendations”. In: *SIGCOMM Comput. Commun. Rev.* 35.3 (July 2005), pp. 19–32. ISSN: 0146-4833. DOI: 10.1145/1070873.1070876. URL: https://doi.org/10.1145/1070873.1070876.

[35] Aaron van den Oord et al. “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499* (2016).

[36] S. Orlowski et al. “SNDlib 1.0–Survivable Network Design Library”. English. In: *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*. Apr. 2007.

[37] Vítor Pereira, Miguel Rocha, and Pedro Sousa. “Traffic Engineering With Three-Segments Routing”. In: *IEEE Transactions on Network and Service Management* 17.3 (2020), pp. 1896–1909. DOI: 10.1109/TNSM.2020.2993207.

[38] Vı´tor Pereira, Miguel Rocha, and Pedro Sousa. “Traffic Engineering with Three-Segments Routing”. In: *IEEE Transactions on Network and Service Management* (2020).

[39] Pinyarash Pinyoanuntapong, Minwoo Lee, and Pu Wang. “Delay-optimal traffic engineering through multi-agent reinforcement learning”. In: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2019, pp. 435–442.

[40] Harald Räcke. “Optimal Hierarchical Decompositions for Congestion Minimization in Networks”. In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. STOC ’08. Victoria, British Columbia, Canada: Association for Computing Machinery, 2008, pp. 255–264. ISBN: 9781605580470. DOI: 10.1145/1374376.1374415. URL: https://doi.org/10.1145/1374376.1374415.

[41] Harald Räcke. “Survey on Oblivious Routing Strategies”. In: *Mathematical Theory and Computational Practice*. Ed. by Klaus Ambos-Spies, Benedikt Löwe, and Wolfgang Merkle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 419–429. ISBN: 978-3-642-03073-4.

[42] Tabish Rashid et al. “Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning”. In: *International conference on machine learning*. PMLR. 2018, pp. 4295–4304.

[43] Jennifer Rexford. “Technical Perspective: Tracking Pandemic-Driven Internet Traffic”. In: *Commun. ACM* 64.7 (June 2021), p. 100. ISSN: 0001-0782. DOI: 10.1145/3465173. URL: https://doi.org/10.1145/3465173.

[44] George F Riley and Thomas R Henderson. “The ns-3 network simulator”. In: *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[45] Timmy Schüller et al. “Failure Resiliency With Only a Few Tunnels – Enabling Segment Routing for Traffic Engineering”. In: *IEEE/ACM Transactions on Networking* 29.1 (2021), pp. 262–274. DOI: 10.1109/TNET.2020.3030543.

[46] Timmy Schüller et al. “Traffic Engineering Using Segment Routing and Considering Requirements of a Carrier IP Network”. In: *IEEE/ACM Transactions on Networking* 26.4 (2018), pp. 1851–1864. DOI: 10.1109/TNET.2018.2854610.

[47] M. Schuster and K. K. Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[48] X. Shi and D. Yeung. “Machine Learning for Spatiotemporal Sequence Forecasting: A Survey”. In: *arXiv preprint arXiv:1808.06865* (2018).

[49] Siva Sivabalan et al. “Pcep extensions for segment routing”. In: *IETF draft-sivabalan-pce-segment-routing-03. txt* (2014).

[50] Stuart Mitchell. *PuLP*. URL: https://coin-or.github.io/pulp/.

[51] Penghao Sun et al. “MARVEL: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning”. In: *Computer Networks* 177 (2020), p. 107230.

[52] Penghao Sun et al. “Sinet: Enabling scalable network routing with deep reinforcement learning on partial nodes”. In: *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*. 2019, pp. 88–89.

[53] Paul Tune and Matthew Roughan. “Spatiotemporal Traffic Matrix Synthesis”. In: *SIGCOMM Comput. Commun. Rev.* 45.4 (Aug. 2015), pp. 579–592. ISSN: 0146-4833. DOI: 10.1145/2829988.2787471. URL: https://doi.org/10.1145/2829988.2787471.

[54] Paul Tune et al. “Internet traffic matrices: A primer”. In: *Recent Advances in Networking* 1 (2013), pp. 1–56.

[55] Victor Heorhiadi. *TMgen*. URL: https://tmgen.readthedocs.io/.

[56] R Vinayakumar, K. P. Soman, and Prabaharan Poornachandran. “Applying deep learning approaches for network traffic prediction”. In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2017, pp. 2353–2358. DOI: 10.1109/ICACCI.2017.8126198.

[57] Jing Wang et al. “Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach”. In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 2017, pp. 1–9. DOI: 10.1109/INFOCOM.2017.8057090.

[58] Zonghan Wu et al. “A Comprehensive Survey on Graph Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24. DOI: 10.1109/TNNLS.2020.2978386.

[59] Zonghan Wu et al. “Graph WaveNet for deep spatial-temporal graph modeling”. In: *International Joint Conference on Artificial Intelligence 2019*. International Joint Conferences on Artificial Intelligence. 2019, pp. 1907–1913.

[60] Junfeng Xie et al. “A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges”. In: *IEEE Communications Surveys Tutorials* 21.1 (2019), pp. 393–430. DOI: 10.1109/COMST.2018.2866942.

[61] K. Xie et al. “Neural Tensor Completion for Accurate Network Monitoring”. In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 2020, pp. 1688–1697.

[62] K. Xie et al. “Sequential and adaptive sampling for matrix completion in network monitoring systems”. In: *IEEE Conference on Computer Communications (INFOCOM’15)*. 2015, pp. 2443–2451.

[63] Kun Xie et al. “Accurate recovery of Internet traffic data: A tensor completion approach”. In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524463.

[64] Kun Xie et al. “Accurate Recovery of Missing Network Measurement Data With Localized Tensor Completion”. In: *IEEE/ACM Trans. Networking* 27.6 (2019), pp. 2222–2235.

[65] S. Xingjian et al. “Convolutional LSTM network: A machine learning approach for precipitation nowcasting”. In: *Advances in neural information processing systems*. 2015, pp. 802–810.

[66] Feiyu Xu et al. “Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges”. In: *Natural Language Processing and Chinese Computing*. Ed. by Jie Tang et al. Cham: Springer International Publishing, 2019, pp. 563–574. ISBN: 978-3-030-32236-6.

[67] Zhiyuan Xu et al. “Experience-driven networking: A deep reinforcement learning based approach”. In: *IEEE INFOCOM 2018-IEEE conference on computer communications*. IEEE. 2018, pp. 1871–1879.

[68] Minghao Ye et al. "DATE: Disturbance-Aware Traffic Engineering with Reinforcement Learning in Software-Defined Networks". In: *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. 2021, pp. 1–10. DOI: 10.1109/IWQOS52092.2021.9521343.

[69] Junjie Zhang et al. "CFR-RL: Traffic Engineering With Reinforcement Learning in SDN". In: *IEEE Journal on Selected Areas in Communications* 38.10 (2020), pp. 2249–2259. DOI: 10.1109/JSAC.2020.3000371.

[70] Yin Zhang. "Abilene Dataset". In: *URL http://www.cs.utexas.edu/yzhang/research/AbileneTM* (2011).

[71] H. Zhou, D. Zhang, and K. Xie. "Accurate traffic matrix completion based on multi-Gaussian models". In: *Computer Communications* 102 (2017), pp. 165–176.

List of publications

Journal papers

1 **Van An Le**, Yusheng Ji, Huu Huy Tran, Phi Le Nguyen, John C. S. Lui, "Achieving Multi-time-step Segment Routing via Traffic Prediction and Compressive Sensing Techniques", IEEE Transactions on Network and Service Management journal, (In submission).

Conference proceedings

1 **Van An Le**, Tien Thanh Le, Phi Le Nguyen, Huynh Thi Thanh Binh, Rajendra Akerkar, Yusheng Ji, "GCRINT: Network Traffic Imputation Using Graph Convolutional Recurrent Neural Network", ICC 2021-IEEE International Conference on Communications (ICC), Jun. 2021. (Full paper, 6 pages).

2 **Van An Le**, Tien Thanh Le, Phi Le Nguyen, Huynh Thi Thanh Binh, Yusheng Ji, "Multi-time-step Segment Routing based Traffic Engineering Leveraging Traffic Prediction", 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), May. 2021. (Full paper, 6 pages).

3 **Van An Le**, Phi Le Nguyen, Yusheng Ji, "Deep Convolutional LSTM network-based Traffic Matrix Prediction with Partial Information", IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Apr. 2019. (Full paper, 9 pages).

Technical reports

1 **Van An Le**, Yusheng Ji, "Dynamic diffusion convolutional recurrent neural network-based traffic prediction", IEICE Technical Report, NS2020-70, pp. 75-80, Oct., 2020 (**Invited Lecture**).

2 **Van An Le**, Yusheng Ji, "Dynamic diffusion convolutional recurrent neural network-based traffic prediction", EICE General Conference 2020 (**English Session Award**).

3 **Van An Le**, Phi Le Nguyen, Yusheng Ji, "Traffic Matrix Prediction based on Bidirectional Recurrent Neural Network and Long Short-Term Memory", IEICE Technical Report, Vol. 118, No. 140, CQ2018-40, pp. 51-56, July 2018.