Predicting Path Failure In Time-Evolving Graphs

Jia Li, Zhichao Han, Hong Cheng, Jiao Su, Pengyun Wang, Jianfeng Zhang, Lujia Pan

The Chinese University of Hong Kong
Noah's Ark Lab, Huawei Technologies

Overview

Graphs are used to model real-world entities and their relationship.

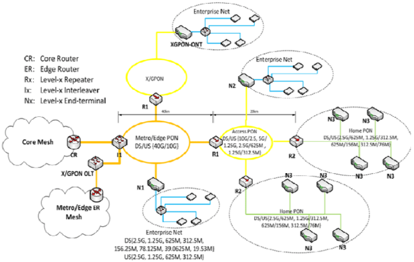

Figure: telecommunication network

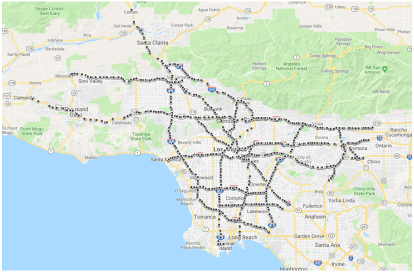

Figure: traffic network

Structure dynamics and temporal dependency

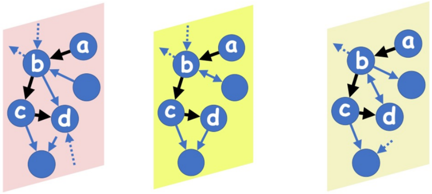

Figure: graph structures are evolving

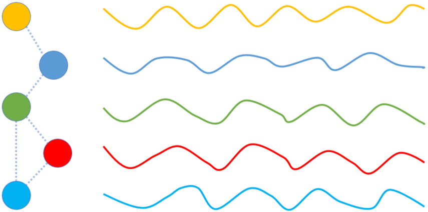

Figure: time series are observed on each node

In this work, we focus on *path classification in a time-evolving graph*, which predicts the status of a path in the near future.

Definition

Time-evolving graph

Denote the adjacency matrix $A^t \in \mathbb{R}^{N \times N}$ and the observed signals $X^t \in \mathbb{R}^{N \times d}$ as a *graph snapshot* at time t, a sequence of graph snapshots over time steps $0, 1, \ldots, t$ is defined as a *time-evolving graph*.

Path availability

Denote a *path* as a sequence $p = \langle v_1, v_2, \ldots, v_m \rangle$ of length m in the time-evolving graph. For the same path, we use $s^t = \langle x_1^t, x_2^t, \ldots, x_m^t \rangle$ to represent the observations of the path nodes at time t. We utilize the past M time steps to predict the availability of this path in the next F time steps.

Example

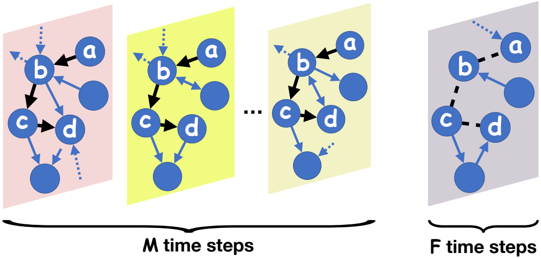

Figure: A time-evolving graph in which four nodes a, b, c, d correspond to four switches in a telecommunication network. Given the observations and graph snapshots in the past M time steps, we want to infer if path $\langle a, b, c, d \rangle$ will fail or not in the next F time steps.

Path classification

We formulate this prediction task as a classification problem and our goal is to learn a function $f(\cdot)$ that can minimize the cross-entropy loss $\mathcal{L}$ over the training set D.

$$\arg\min \mathcal{L} = -\sum_{\mathbf{P}_j \in D} \sum_{c=1}^{C} Y_{jc} \log f_c(\mathbf{P}_j), \quad (1)$$

where $\mathbf{P}_j = ([s_j^{t-M+1}, \ldots, s_j^t], p_j, [A^{t-M+1}, \ldots, A^t])$ is a training instance, $Y_j \in \{0,1\}^C$ is the training label representing the availability of this path in the next F time steps, $f_c(\mathbf{P}_j)$ is the predicted probability of class c, and C is the number of classes.

Three properties

- **Node correlation**: Observations on nodes are correlated;
- **Graph structure dynamics**: Observations on nodes are influenced by the changes on the graph structure;
- **Temporal dependency**: The time series recorded on each node demonstrates strong temporal dependency.

Framework

Our model uses a two-layer LRGCN, to obtain the hidden representation of each node. Then it utilizes a self-attentive mechanism to learn the node importance and encode it into a unified path representation.

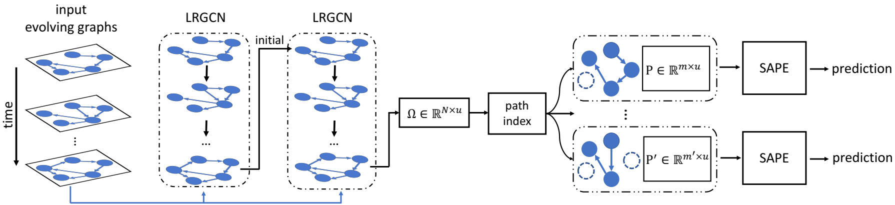

Figure: Framework of the proposed model for path classification.

Static graph modeling

Relational GCN (R-GCN) by Kipf et al. is developed to deal with multi-relational static graphs.

R-GCN

$$Z = \sigma(\sum_{\phi \in R} (D_{\phi}^{t})^{-1} A_{\phi}^{t} X^{t} W_{\phi} + X^{t} W_0), \tag{2}$$

$R = \{in, out\}$. $(D_{\phi}^{t})_{ii} = \sum_j (A_{\phi}^{t})_{ij}$.
$A_{in}^{t} = A^{t}$ represents the incoming relation.
$A_{out}^{t} = (A^{t})^{T}$ represents the outgoing relation.

We view the effect of self-connection normalization as a linear combination of incoming and outgoing normalization.

Simplified R-GCN

$$Z_s = \sigma(\sum_{\phi \in R} \tilde{A}^t_\phi X^t W_\phi), \tag{3}$$

where $\tilde{A}^t_\phi = (\hat{D}^t_\phi)^{-1} \hat{A}^t_\phi$. $\hat{A}^t_\phi = A^t_\phi + I_N$. $(\hat{D}^t_\phi)_{ii} = \sum_j (\hat{A}^t_\phi)_{ij}$.

Two-hop simplified R-GCN

$$\Theta_s \star g \; X^t = \sum_{\phi \in R} \tilde{A}_\phi^t \sigma\Big(\sum_{\phi \in R} \tilde{A}_\phi^t X^t W_\phi^{(0)}\Big) W_\phi^{(1)}. \tag{4}$$

where Θ_s represents the parameter set used in the static graph modeling, $W_\phi^{(0)} \in \mathbb{R}^{d \times h}$ is an input-to-hidden weight matrix for a hidden layer with h feature maps. $W_\phi^{(1)} \in \mathbb{R}^{h \times u}$ is a hidden-to-output weight matrix, $\star g$ stands for this two-hop graph convolution operation and shall be used thereafter.

Adjacent graph snapshots modeling

Before diving into a sequence of graph snapshots, we first focus on two adjacent time steps $t-1$ and t.

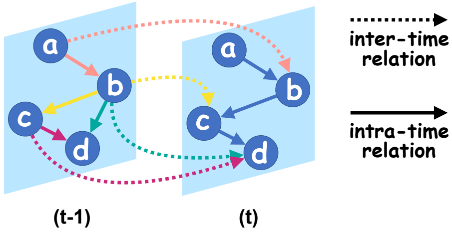

Figure: Plot of intra-time relation (in solid line) and inter-time relation (in dotted line) modeled for two adjacent graph snapshots.

There are four types of relations, i.e., intra-incoming, intra-outgoing, inter-incoming and inter-outgoing relations.

This operation is named time-evolving graph **G_unit**, which has a similar role of unit in RNN.

time-evolving graph unit

$$G\_unit(\Theta, [X^t, X^{t-1}]) = \sigma(\Theta_s \star g\ X^t + \Theta_h \star g\ X^{t-1}). \tag{5}$$

where Θ_h stands for the parameter set used in inter-time modeling, and it does not change over time. For $\Theta_h \star g\ X^{t-1}$, $\tilde{A}_{\phi}^{t-1}$ is used to represent the graph structure.

The proposed LRGCN model

We first design a RNN-style neural network working on a time-evolving graph.

$$H^t = \sigma(\Theta_H \star g \; [X^t, H^{t-1}]). \tag{6}$$

where Θ_H includes Θ_s and Θ_h.

We propose a Long Short-Term Memory R-GCN. LRGCN utilizes three gates to achieve the long-term memory or accumulation.

$$\mathbf{i}^t = \sigma(\Theta_i \star g \, [X^t, H^{t-1}]) \tag{7}$$

$$\mathbf{f}^t = \sigma(\Theta_f \star g \, [X^t, H^{t-1}]) \tag{8}$$

$$\mathbf{o}^t = \sigma(\Theta_o \star g \, [X^t, H^{t-1}]) \tag{9}$$

$$\mathbf{c}^t = \mathbf{f}^t \odot \mathbf{c}^{t-1} + \mathbf{i}^t \odot \tanh(\Theta_c \star g \, [X^t, H^{t-1}]) \tag{10}$$

$$H^t = \mathbf{o}^t \odot \mathbf{c}^t \tag{11}$$

where $\odot$ stands for element-wise multiplication, $\mathbf{i}^t$, $\mathbf{f}^t$, $\mathbf{o}^t$ are input gate, forget gate and output gate at time t respectively.

Two challenges

For the final path classification task, however, we still identify several challenges:

- **Size invariance**: How to produce a fixed-length vector representation for any path of arbitrary length?
- **Node importance**: How to encode the importance of different nodes into a unified path representation?

SAPE

We propose a self-attentive path embedding method, called SAPE, to address the challenges listed above.

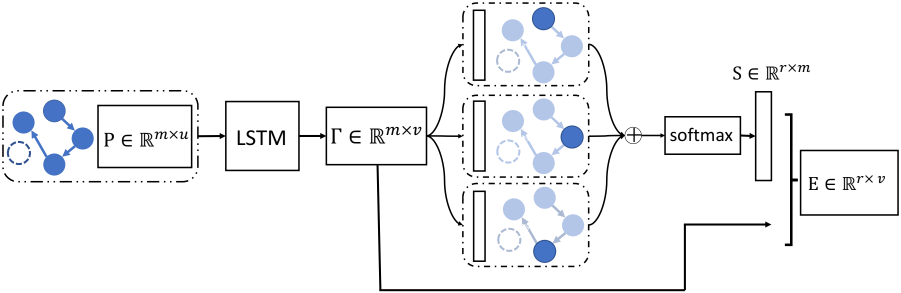

Figure: The proposed self-attentive path embedding method SAPE.

In SAPE, we first utilize LSTM to sequentially take in node representation of a path. Then we use the self-attentive mechanism to learn the node importance and transform a path of variable length into a fixed-length embedding vector.

$$\Gamma = \text{LSTM}(P) \tag{12}$$

$$S = \text{softmax}\big(W_{h2}\tanh(W_{h1}\Gamma^T)\big) \tag{13}$$

$$E = S\Gamma \tag{14}$$

where $\Gamma \in \mathbb{R}^{m\times v}$. $W_{h1} \in \mathbb{R}^{d_s\times v}$ and $W_{h2} \in \mathbb{R}^{r\times d_s}$ are two weight matrices. E is size invariant since it does not depend on the number of nodes m.

We validate our model on two real-world data sets: (1) predicting path failure in a telecommunication network, and (2) predicting path congestion in a traffic network.

Table: Statistics of path instances

	Telecom	**Traffic**
No. of failure/congestion	385,896	85,083
No. of availability	6,821,101	346,917
Average length of paths	7.05 ± 4.39	32.56 ± 12.48

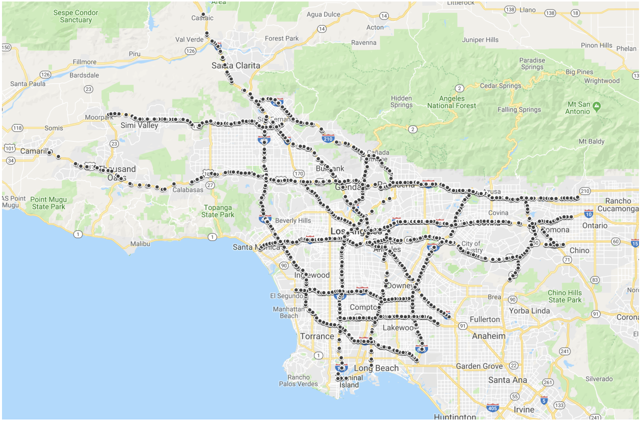

Figure: Sensor distribution in District 7 of California. Each dot represents a sensor station.

- DTW, does not use graph structure.
- FC-LSTM, does not use graph structure.
- DCRNN, it works on a static graph.
- STGCN, it works on a static graph.
- LRGCN, it works on a static graph.
- LRGCN-SAPE (static), which is similar to LRGCN except that we replace the path representation method LSTM with SAPE.
- LRGCN-SAPE (evolving), which is similar to LRGCN-SAPE (static) except that the underlying graph structure evolves over time.

Table: Comparison of different methods on path failure prediction on Telecom

	Algorithm	Precision	Recall	Macro-F1
1	**DTW**	15.47%	9.63%	53.23%
2	**FC-LSTM**	13.29 %	52.27 %	53.78 %
3	**DCRNN**	13.97 %	57.81 %	54.42 %
	STGCN	16.35 %	52.53 %	56.29 %
	LRGCN	17.38 %	61.34 %	57.70 %
4	**LRGCN-SAPE (static)**	17.67 %	**65.28** %	60.55 %
	LRGCN-SAPE (evolving)	**19.23** %	65.07 %	**61.89** %

Table: Comparison of different methods on path congestion prediction on Traffic

	Algorithm	Precision	Recall	Macro-F1
1	**DTW**	12.05%	39.12%	51.62%
2	**FC-LSTM**	54.44 %	87.97 %	76.55 %
3	**DCRNN**	63.05 %	**88.55** %	82.60 %
	STGCN	64.52 %	86.15 %	82.41 %
	LRGCN	65.15 %	87.65 %	83.74 %
4	**LRGCN-SAPE (static)**	67.74 %	88.44%	84.84 %
	LRGCN-SAPE (evolving)	**71.04** %	88.50 %	**86.74** %

Benefits of graph evolution modeling

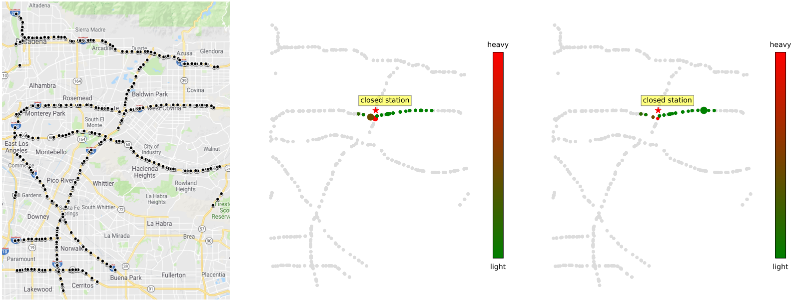

Figure: Visualization of learned attention weights of a path on Traffic (left: the original map; middle: attention weights by LRGCN-SAPE (evolving); right: attention weights by LRGCN-SAPE (static)).

Training efficiency

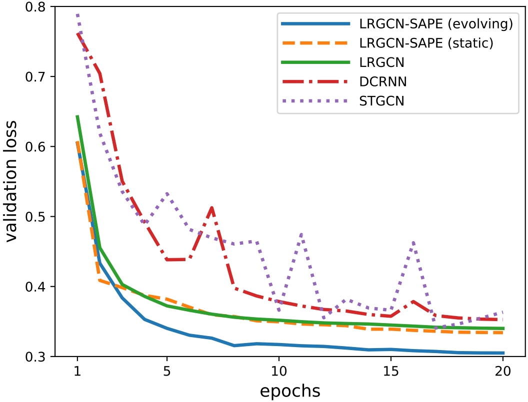

Figure: Learning curve of different methods. LRGCN-SAPE (evolving) achieves the lowest validation loss.

Path embedding visualization

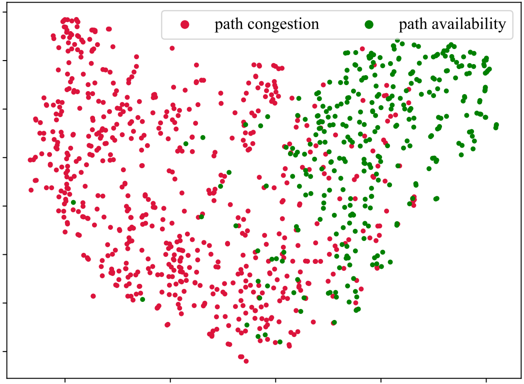

Figure: Two-dimensional visualization of path embeddings on Traffic using SAPE.

Conclusion

- We study path classification in time-evolving graphs.
- We design a new dynamic graph neural network LRGCN, which views node correlation within a graph snapshot as intra-time relations, and views temporal dependency between adjacent graph snapshots as inter-time relations.

Data and code:

https://github.com/chocolates/Predicting-Path-Failure-In-Time-Evolving-Graphs

Thank you.