Self-Supervision on Dynamic Graphs

Yusong Ye 004757800
yusongye@g.ucla.edu

ABSTRACT

Graph Neural Network becomes extremely popular recently. It works well in handling graph related data by learning the node representations. However, in real-world scenario, graph structure may not be fully available. Therefore, there are some research trying to figure out a way to learn the latent graph structure. Traditional ways to do this including using similarity graph and link nodes with similar attributes, leveraging domain knowledge to infer the graph structure. Recent literature focus on using graph neural networks to do the latent graph structure learning using trainable graph generator[8].

In our work, we want to consider a more interesting and realistic setting, handling graph related task in a dynamic graph setting where the graph structure is not readily available. Our experiment results shows that even without little knowledge from the graph structure, we can achieve good performance on the latent structure generation and downstream tasks.

KEYWORDS

graph neural networks, self-supervision, dynamic graphs

ACM Reference Format:
Yusong Ye 004757800. 2018. Self-Supervision on Dynamic Graphs. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Nowadays, with graph neural networks becoming more and more popular, many people consider applying it to real-world datasets. Traditional graph related data, like non-dynamic, non-knowledge based data-sets are learned very well by the powerful graph embeddings. However, in the real-world case, most datasets are dynamic and changing according to time. Think about this example, you want to predict the traffic given historical data, what should you do? The speed, which may be represented in graph as node values are changing over time and you cannot treat it like a static graph.

Therefore, dynamic graph and its related GNNs are very worth discussing. In our work, we consider a case when dynamic graph structures are not readily available. We want to learn the downstream task and the graph structure at the same time. The main difficulty in doing this is that the model capacity is very large. There are a lots of parameters to be tuned at the same time. Therefore, we need to find an efficient way to train it.

For the detailed method, we will talk more in the later sections. Note that for the dataset, we consider generated opinion migration dataset[2]. Basically, the nodes are representing users, and the features are their opinion. The changing graph structure representing the changing of their relationship on social media.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 PROBLEM DEFINITION

Given a set of node N, and its correlated features F, and time span from $T = 1$ to $T = n$. We want to predict features at $T = n + 1$. Also, we want to learn the latent graph structures.

3 RELATED WORK

3.1 Graph Neural Networks

During the past years, graph neural networks(GNNs) attracts the great attention and has been widely used in solving machine learning problems on graph data. Due to the well performance, GNNs have been applied on various analytical tasks.[5] There are mainly three different type tasks for GNNs, the node-level tasks, edge-level tasks and graph-level tasks. Node-level tasks focus on categorizing nodes into different classes, predict a continuous value for each node and partitioning the nodes into several disjoint groups where similar nodes should be in the same group. Edge-level tasks focus on classifying edge types and predicting whether there is an edge existing between two given nodes. Graph-level tasks are all about learning graph representation and figure out graph classification or graph matching problems. [6] Dynamic Neural networks can be considered as the improvement of the static neural networks in which by adding more decision algorithms we can make neural networks learning dynamically from the input and generate better quality results.

3.2 Graph Structural Learning

Although promising results have been achieved, recent studies have shown that GNNs are vulnerable to adversarial attacks which means the performance of GNNs can dramatically decreased under an unnoticeable perturbation in graphs. Noisy or incomplete graphs often lead to unsatisfactory graph representations and result in wrong predictions. Therefore, graph structure learning(GSL) is proposed to learn an optimized graph structure and corresponding graph representations for downstream tasks. For instance, a general framework Pro-GNN has been proposed to jointly learn a structural graph and a robust graph neural network model from the perturbed graph. [3] According to the experiments of Pro-GNN on Cora, Citeseer, Polblogs and Pubmed datasets, the model improves the overall node classification performance about 5% to 15% under various adversarial attacks.

3.2.1 GSL Pipeline. There are mainly three parts for most of current GSL models which is shown in Figure 1 above, including graph

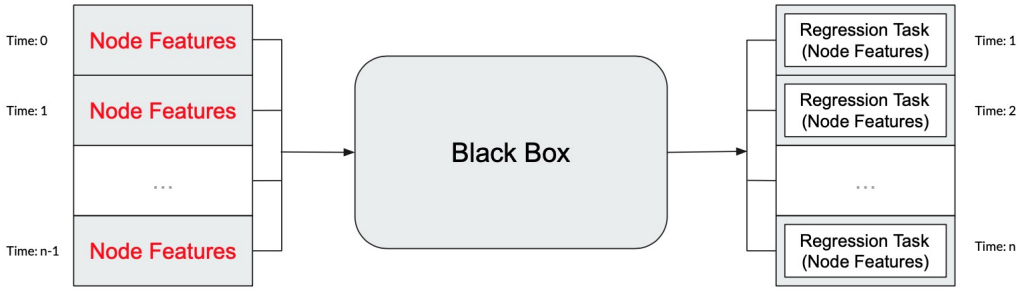

Figure 1: Problem Formulation

construction, graph structure modeling and message propagation. A graph is composed of vertices and edges as $G = (V, E)$ where V is a set of nodes and E is a set of edges. There are two main approaches k Nearst Neighbors which connecting k-closest neighbors of each vertex, and ϵ proximity thresholding that create an edge between two nodes if their similarity is smaller than a preset threshold ϵ.[?]

3.2.2 Approaches of Structure Modeling. The establishment of structure learner g is the key task of graph structure modeling process. Learner g models the edge connectivity to refine and optimize the preliminary graph by an encoding function. Currently, we categorize the existing studying into the following three methods, metric-based approaches, neural approaches and direct approaches.

Metric-based approaches applied kernel functions on pairwise node embeddings to compute the similarity between node features and derive edge weights. This approach refine the graph structure by promoting the intra-class connections, leading to more compact representations and could help improve the capability of end-to-end training. Neural approaches infer edge weights by leveraging more expressive neural networks.Direct approaches directly optimize the adjacency matrix as learnable parameters with GNN.

3.2.3 Message Propagation. A GNN is composed of several message propagation layers. We define there are K layers and N_n as a set of neighbors of node v_n. The propagation process of the k-th layer in GNN is two steps: [4]

$$\mathbf{m}_{k,n} = \text{AGGREGATE}\left(\left\{\mathbf{h}_{k-1,u} : u \in \mathcal{N}(n)\right\}\right)$$
$$\mathbf{h}_{k,n} = \text{UPDATE}\left(\mathbf{h}_{k-1,n}, \mathbf{m}_{k,n}, \mathbf{h}_{0,n}\right)$$

AGGREGATE is a permutation invariant function. GNN follow this message propagation form with different aggregate function and update function after K message-passing layers.

3.3 SLAPS

3.3.1 SLAPS. SLAPS [1] (Simultaneous Learning of Adjacency and GNN Parameters with Self-supervision) proposed a supplement task to the training of graph generator with a self-supervised objective to increase the amount of supervision in learning a structure. The goal of SLAPS is to learn latent graph representation so that the performance on downstream tasks like node classification can be improved. It is composed of four main components: generator,adjacency processor,classifier, and self-supervision.

3.3.2 Generator. The Generator is a function $\mathrm{G} : \mathbb{R}^{n\times f} \to \mathbb{R}^{n\times n}$ with parameters θ_G which takes the node features $\boldsymbol{X} \in \mathbb{R}^{n\times f}$ as input and produces a matrix $\tilde{\boldsymbol{A}} \in \mathbb{R}^{n\times n}$ as output. There are mainly three kinds of generators:

- Full Parameterization(FP): $\tilde{\boldsymbol{A}} = \mathrm{G}_{FP}\left(\boldsymbol{X}; \boldsymbol{\theta}_{\mathrm{G}}\right) = \boldsymbol{\theta}_{\mathrm{G}}$, which means the generator ignores the input node features and produces a matrix $\tilde{\boldsymbol{A}}$ as output.
- MLP-kNN: $\tilde{\boldsymbol{A}} = \mathrm{G}_{MLP}\left(\boldsymbol{X}; \boldsymbol{\theta}_{\mathrm{G}}\right) = kNN\left(MLP\left(\boldsymbol{X}\right)\right)$. MLP : $\mathbb{R}^{n\times f} \to \mathbb{R}^{n\times f'}$ is an MLP $\mathbb{R}^{n\times f} \to \mathbb{R}^{n\times f'}$ that produces a matrix with updated node representations $\boldsymbol{X'}$ and kNN $\mathbb{R}^{n\times f'} \to \mathbb{R}^{n\times n}$ produces a sparse matrix where each node has up to k neighbors. Final output can be written as $\widetilde{\boldsymbol{A}}_{ij} = \phi\left(\boldsymbol{z}_i, \boldsymbol{z}_j\right)$, where $\boldsymbol{z}_i$ represents the i-th row of the output of MLP. Here the metric function ϕ is simply the cosine similarity.

3.3.3 Adjacency Processor. The Adjacency Processor is a function $\mathrm{G} : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ which forces the adjacency matrix to be positive, symmetric and normalized. We use $\boldsymbol{A} = \frac{1}{2}\boldsymbol{D}^{-\frac{1}{2}}\left(\mathrm{P}(\tilde{\boldsymbol{A}}) + \mathrm{P}(\tilde{\boldsymbol{A}})^T\right)\boldsymbol{D}^{-\frac{1}{2}}$, where P is a function with a non-negative range applied element-wise on its input. D is the degree matrix of $\frac{1}{2}\left(\mathrm{P}(\tilde{\boldsymbol{A}}) + \mathrm{P}(\tilde{\boldsymbol{A}})^T\right)$ and guarantee the normalization.

3.3.4 classifier. The classifier is a function $\mathrm{GNN_c} : \mathbb{R}^{n\times f} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times|C|}$. C represents classes and |C| represents the number of classes. Its parameter $\theta_{\mathrm{GNN_c}} = \{\boldsymbol{W}^1, \boldsymbol{W}^2\}$ generated by a two layers GNC(Graph Convolutional Network). The function is defined as $\mathrm{GNN}_c(\boldsymbol{A}, \boldsymbol{X}; \theta_{\mathrm{GNN_c}}) = \boldsymbol{A}\mathbb{RLU}(\boldsymbol{AXW}^{(1)})\boldsymbol{W}^2$. The training loss L_C for the classification task is computed by take the softmax of the logitis to produce a probability distribution for each node and then computing the cross-entropy loss.

3.3.5 self-supervision. Self-supervised learning (SSL) is a method of machine learning that learns from unlabeled sample data by artificially initializing network weights. The self-supervised task of SLAPs work is based on denoising autoencoder. It adds noisy feature to node feature, simultaneously train two GNN model with and without noisy feature and compare the results.

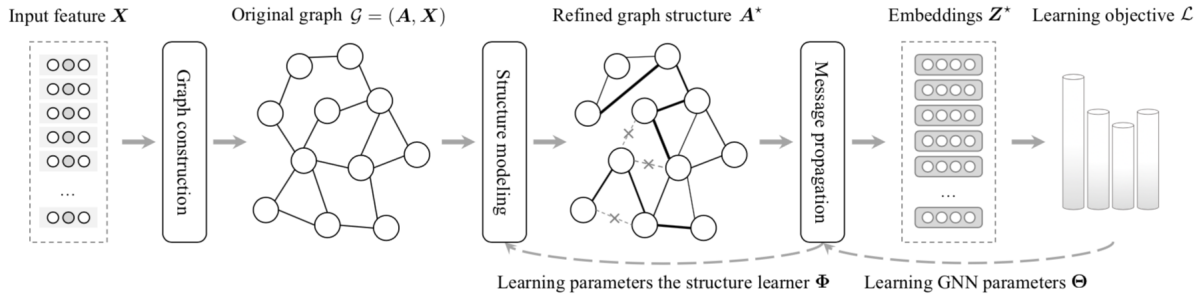

Figure 2: General Pipeline of GSL[?]

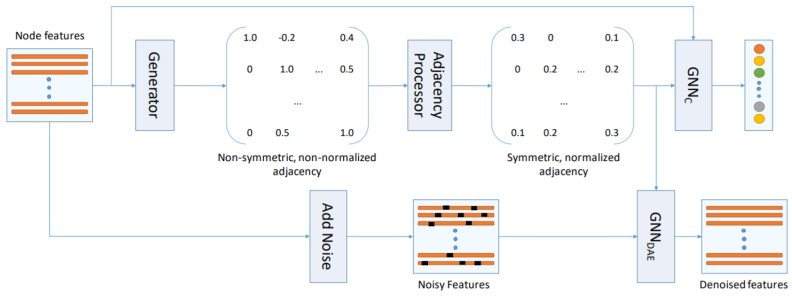

Figure 1: Overview of SLAPS. At the top, a generator receives the node features and produces a non-symmetric, non-normalized adjacency having (possibly) both positive and negative values (Section 4.1). The adjacency processor makes the values positive, symmetrizes and normalizes the adjacency (Section 4.2). The resulting adjacency and the node features go into GNN_C which predicts the node classes (Section 4.3). At the bottom, some noise is added to the node features. The resulting noisy features and the generated adjacency go into GNN_DAE which then denoises the features (Section 4.5).

Figure 3: Overview of SLAP [1]

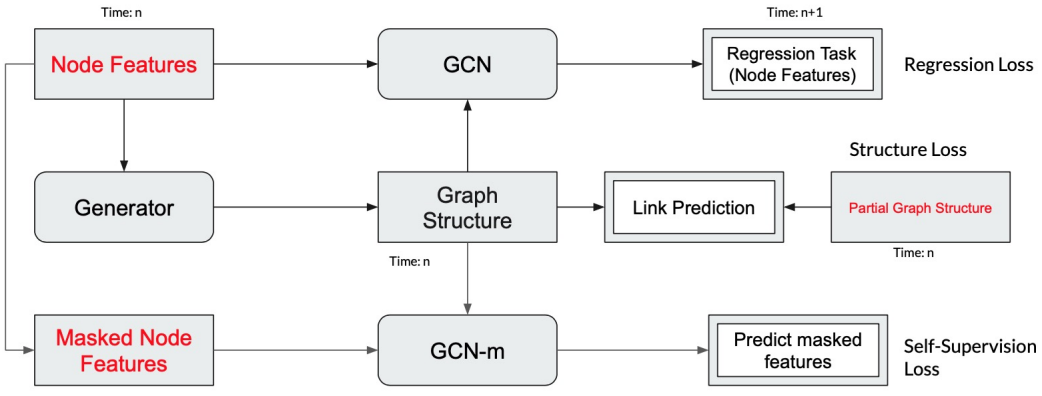

Figure 4: Framework

4 OUR METHOD

From Figure 4, we can see that there are three loss functions, which is the key part of our method. At time $T = n$, we have node features F and we pass the features to a *GCN* that predicts the node features at next time span $T = n + 1$. At the same time, we use a graph generator to generate the graph structure based on the node features. We give *GCN* our trained graph structure and node features and let it do prediction. Note that regression loss can tune the *GCN* and *Generator* at the same time, but there are too many parameters, and it is better if we could use more loss to guide the learning of *GCN* and *Generator*.

4.1 Structure loss

Structure loss is used to guide the generation of graph structure. We provide partial structure information (we try different partial structure in the experiment setting) to the generator to guide it to generate more reasonable graph structures. There are two kinds of generator that we use in this work. Note that the self-adaptive adj is initially purely parameter based, and I concat the node feature with the learnable embeddings to let it leverage the feature information

4.2 Self-Supervision Loss

For the self-supervision loss, we want it to guide the performance increase on both structure learning task and the downstream task. The idea is that the structure is closely related to the node features. In this way, if we make the self-supervision task more tricky, the model can learn better. The idea is that at each time span $T = n$, we mask out entire node features and give them special mask. We give this masked out node features to another *GCN* to predict the masked out node features.

5 EXPERIMENT

5.1 Dataset

We want to use the opinion migration dataset as discussed above, we generate small scale datasets, with node size varying from 25, 50 to 100. Figure6 is a visualization of the adjacency matrix.

5.2 Experiment Results

From Figure7, we can see that both self-supervision and structure loss can help us guide the learning process of the graph structure and downstream task. Even with limited structure ground truth, we can learn very promising graph structure.

5.3 Case Study

From 8, we can see that the graph structure we generate with only 5% groundtruth label is very powerful.

6 FUTURE WORK

I am currently working on applying this model to large scale, which is more realistic but challenging. The idea is to do the learning on a small sampled graph and apply it to a larger graph.

Also, I am testing this model on real-world datasets.

REFERENCES

[1] Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. 2021. SLAPS: Self-Supervision Improves Structure Learning for Graph Neural Networks. *CoRR* abs/2102.05034 (2021). arXiv:2102.05034 https://arxiv.org/abs/2102.05034

[2] Yupeng Gu, Yizhou Sun, and Jianxi Gao. 2017. The Co-Evolution Model for Social Network Evolving and Opinion Migration. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. Association for Computing Machinery, New York, NY, USA, 175–184. https://doi.org/10.1145/3097983.3098002

[3] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 66–74.

[4] Teng Xiao, Zhengyu Chen, Donglin Wang, and Suhang Wang. 2021. Learning how to propagate messages in graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1894–1903.

[5] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[6] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81.

[7]]zhusurvey Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Yuanqi Du, Jieyu Zhang, Qiang Liu, Carl Yang, and Shu Wu. [n. d.]. A Survey on Graph Structure Learning: Progress and Opportunities. ([n. d.]).

[8] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang. 2021. Deep Graph Structure Learning for Robust Representations: A Survey. *CoRR* abs/2103.03036 (2021). arXiv:2103.03036 https://arxiv.org/abs/2103.03036

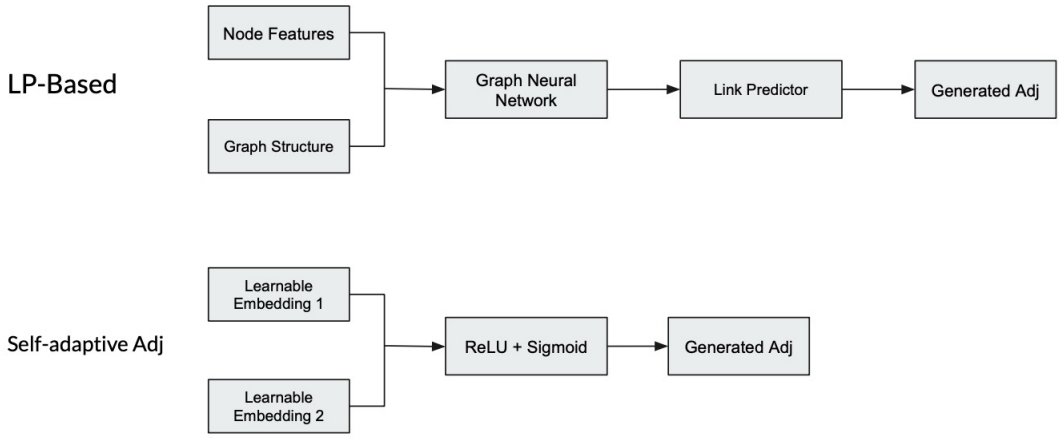

Figure 5: Generator

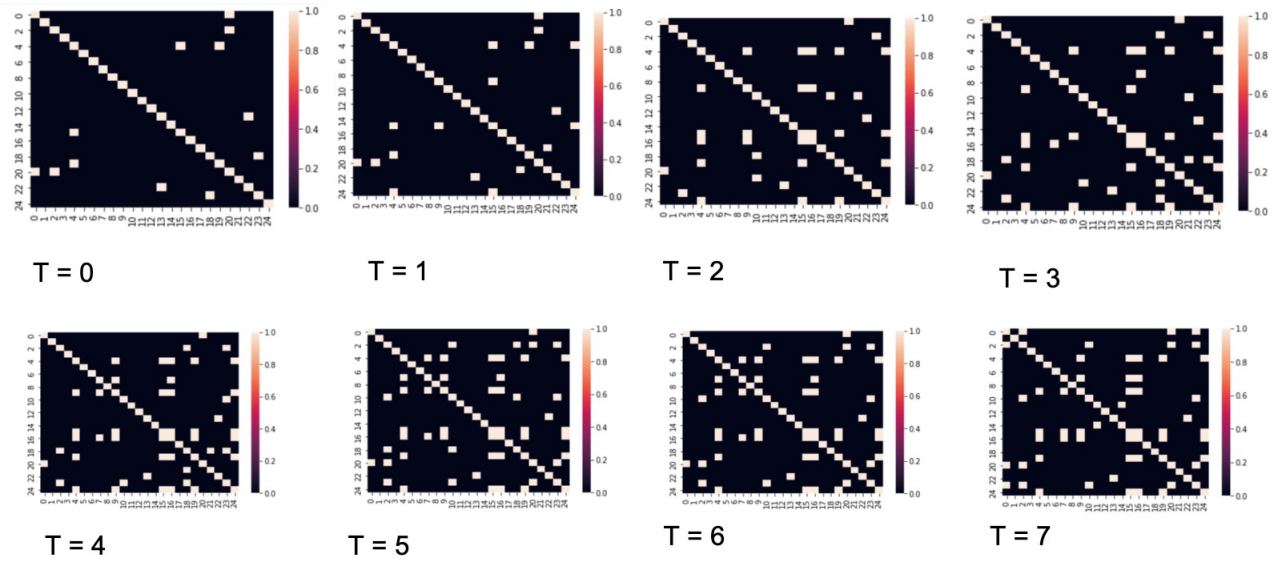

Figure 6: Dataset Visualization

TimeSpan=10 Node num = 50 Percentage of Partial structure Ground truth(%)	Random Structure + 400 epochs Regression Loss	100 epochs Structure Loss + 300 epochs Regression Loss	100 epochs Structure Loss + 300 epochs (Regression Loss + Self-supervision 10% masked)	100 epochs Structure Loss + 300 epochs (Regression Loss + Self-supervision 20% masked)	100 epochs Structure Loss + 300 epochs (Regression Loss + Self-supervision 50% masked)
0%	20.98 / 33%	2.18 / 42%	2.91 / 43%	2.99 / 67%	3.6 / 53%
5%	-	2.91 / 49%	2.87 / 48%	2.70 / 55%	3.17 / 55%
20%	-	2.72 / 62%	2.73 / 55%	1.9 / 72%	2.8 / 81%
50%	-	2.72 / 64%	2.85 / 67%	1.62 / 81%	1.34 / 88%
100%	-	2.45 / 63%	2.32 / 72%	1.42 / 89%	1.32 / 92.6%

Figure 7: Experiment Result

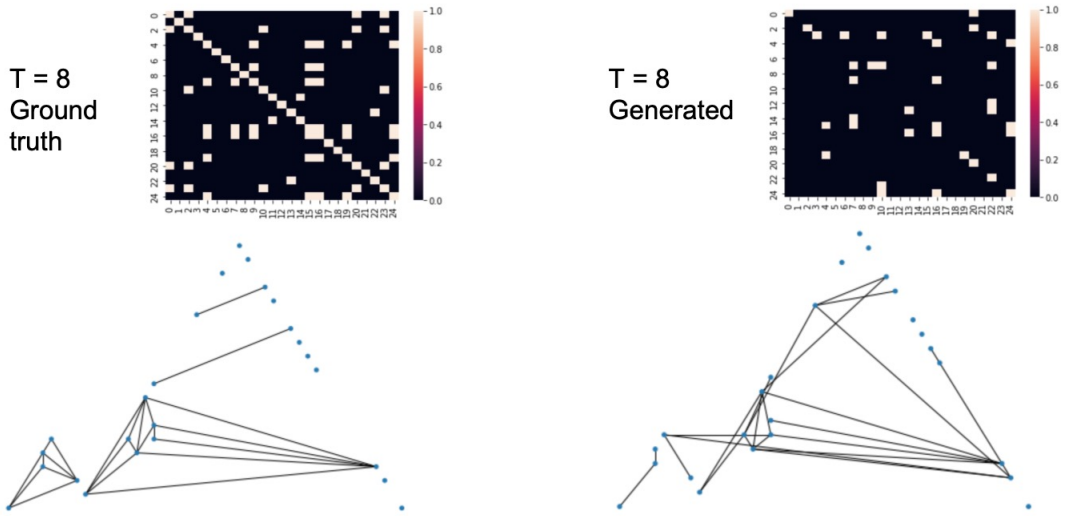

Figure 8: Case Study