Distribution Agreement

Signature:

__________________________ ____________

Yibo Wang Date

Generating Traces of Application Behavior Using Generative Adversarial Networks

By

Yibo Wang
Master of Science

Department of Computer Science

Arnold Dorian, Ph.D.
Advisor

Liang Zhao, Ph.D.
Committee Member

Vaidy Sunderam, Ph.D.
Committee Member

Accepted:

Lisa A. Tedesco, Ph.D.
Dean of the Graduate School

Date

Generating Traces of Application Behavior Using Generative Adversarial Networks

by

Yibo Wang
B.S., Shandong University

Advisor : Arnold Dorian, Ph.D.

Abstract of
A Thesis submitted to the Faculty of the Graduate School
of Emory University in partial fulfillment
of the requirements of the degree of
Master of Science

Department of Mathematics and Computer Science

2021

Abstract

Generally, applications, benchmarks and proxy applications are used for performance analysis of high-performance computing systems. These system performance analysis methods can be challenging or difficult to use, and often application traces are used as workload substitutes. But collecting these traces can also be difficult or time consuming. Therefore, we study synthetic trace generation to synthesize application trace that are indistinguishable from real traces. We propose a machine learning based synthetic data generation method that utilizes temporal graph generative adversarial networks (TG-GANs). We consider communication traces as temporal directed graphs with edge attributes and adjust TG-GAN to generate synthetic data. We use real traces as inputs and generate synthetic ones in some selected representative time windows and evaluate the quality of synthetic data using both quantitative metrics and visualizations. Visualization and quantitative results show that TG-GAN has the potential to generate high-quality synthetic traces but also has some limitations.

Generating Traces of Application Behavior Using Generative Adversarial Networks

by

Yibo Wang
B.S., Shandong University, 2019

Advisor : Arnold Dorian, Ph.D.

A Thesis submitted to the Faculty of the Graduate School
of Emory University in partial fulfillment
of the requirements of the degree of
Master of Science

Department of Mathematics and Computer Science

2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Application workloads, including full applications, application proxies and application traces or profiles, are usually utilized to assess the performance of hardware and software systems. However, these traditional performance assessment methods have various limitations. For example, full applications may be inaccessible. For one thing, some applications are hard to build and run from scratch. For the other thing, some applications are complex to understand, classified or export controlled. Also, applications can require long run time and resources at scale. Application proxies including benchmarks are simpler to build, run and understand, but still require long run time and resources. Similarly, collecting real traces or profiles from real application instances also require long run time and resources. To address these challenges, we propose to generate synthetic traces which are representative of those collected from real application instances using machine learning methods.

Particularly, we use a machine learning model called generative adversarial networks (GANs), which is proposed by [7], is a framework of machine learning methods. Typically, GAN has a generative network to generate synthetic data and a discriminative network to evaluate data. The training objective of the generative network is to generate novel synthetic data which will be evaluated as true data by the discriminative network, while the training objective of the discriminative network is to distinguish synthetic data from true data. The contest between the generative network and the discriminative network helps both networks obtain better performance. GANs have been proven to be a powerful method to generate high-quality synthetic data.

[16] is a robust approach to generate directed temporal graphs. TG-GAN can generate time stamp, node, and edge information for random walks and then assemble random walks to temporal graphs under the learned topological and temporal dependencies. TG-GAN is well-suited for our problem formulation, but requires some adaptations as described in Section 3.3.2.

1.2 Synthetic Trace Generation

Thesis Statement: We can use Machine Learning, especially GANs (Generative Adversarial Networks), to generate traces that are representative of those collected from real application runs.

We consider communication traces as temporal directed graphs with edge attributes, and adjust temporal graph generative adversarial networks (TG-GAN) [16] to generate synthetic data. To make TG-GAN more in line with our problem, we adjust the model to generate [send time, receive time, send node, receive node, data type, data count] instead of [time stamp, send node, receive node] in TG-GAN. Generating synthetic communication traces is a great way to reduce running time and resources consumed by generating real traces. GANs have also been successfully applied to many different domains to generate high-quality synthetic data. For example, [3] proposed medical Generative Adversarial Network (medGAN) to generate high-dimensional, multi-label discrete variables that represent events in EHRs. TG-GAN is a strong approach for generating temporal graphs. TG-GAN is almost perfectly suitable for our case, except for the following points. First, TG-GAN assumes that there is no time interval between send and receive, but in our case, the time interval is very important information. Second, application communication profiles carry more information, like data type and size, than TG-GAN can generate.

Also, we evaluate our approach by comparing the visualized synthetic data and the corresponding real data in different time windows. And we use the comparison of three continuous discrete-time graphs of both synthetic and real data to verify the performance of generated temporal features in synthetic data.

1.3 Contribution and Structure

Our contributions are:

- an adaptation of the TG-GAN model to generate [send time, receive time, send node, receive node, data type, data count] instead of [time stamp, send node, receive node] in TG-GAN.

- a method to generate synthetic application traces for system performance analysis, which is something no one has been done before.

The rest of this paper is organized as follows. In chapter 2, we first introduce previous research on data generation, generative adversarial networks and temporal-graph generative adversarial networks. Chapter 3 illustrates detailed methods and experiments, including data preprocessing, our generative model and evaluation methods. Then in the chapter 4, we present and analyze the results of our experiments. In chapter 5 we summarize our work and discuss potential improvements and future work.

Chapter 2

Background and Related Work

2.1 Data Generation

Many deep generative models have been proposed during the past years. Autoregressive model, which is one of the fully observed likelihood-based deep generative models, is easy to train and evaluate likelihoods; however, it cannot learn features in an unsupervised way. Latent variable models, like variational autoencoder[12], are natural for unsupervised learning tasks and easy to define a complex model using simple building blocks; however, they are hard to evaluate likelihood and the posterior inference is hard. Generative adversarial networks, an implicit generative model, are unsupervised and easy to train. Generative adversarial networks are the state-of-the-art generative models and have been proven to be effective and powerful in various problems and domains, such as medical, biology and social science.

Many deep graph generation methods have been proposed based on the above generative models. For example, GraphVAE [14] is one kind of variatinal autoencoders suitable for generating small graphs; NetGAN [2] is based on GANs framework and generates synthetic random walks; GraphRNN [15] is a deep autoregressive model and learns to generate graphs by training on a representative set of graphs and generates a sequence of node and edge. All these approaches are used to generate static graphs, while for generating dynamic graphs, very few methods have been raised. TagGen [17] generates graphs preserving both the structural and temporal characteristics of the real data; however, it does not handle continuous-time temporal graphs and time validity constraints, while TG-GAN [16] has the ability to generate continuous-time temporal graph with temporal validity. Considering our problem formulation, we choose TG-GAN as the most appropriate deep graph generative approach.

2.2 Generative Adversarial Networks

Generative adversarial networks are very powerful generation frameworks. GANs, which is defined by

$$min_G max_D L_{GAN}(D, G) = E_{X \sim P_{data(x)}}[logD(x) + E_{Z \sim P_{Z(Z)}}[log(1 - D(G(z))]], \tag{2.1}$$

simultaneously train two models, a discriminator model D and a generative model G. X is real data while Z is synthetic data. The discriminator is used to determine whether the input example is from real data or not, while the generator is used to generate synthetic data that will not be detected by the discriminator. In other words, the generator is to maximize the probability of the discriminator making a mistake. This framework is equivalent to a minmax two-player game.

In particular, the generator generates new instances from scratch, rather than sample from real data. This feature allows GANs to generate brand new samples instead of simply sampling from real data, making the generated date more complex and avoiding duplications. Then the contest between the generative model and the discriminative model helps both models obtain better performance. Finally, the generator is gradually corrected until has the ability to generate high-quality data similar as real data.

2.3 Temporal Graph Generative Adversarial Network

Temporal graph generative adversarial network, also known as TG-GAN, is a GAN framework for continuous-time graph generation with time-validity constraints.

In TG-GAN, the generator G utilizes LSTM [10] structure, which is an artificial recurrent neural network (RNN) architecture using feedback connections, as basic model, and its output is a sequence of special temporal walks. The discriminator D has a similar LSTM classifications. And TG-GAN also uses samplers to extract time budget temporal walk from all the graphs. We can make use of the sampler parameter to fine-tune the density of the generated graphs.

Temporal graphs have many different features from static graphs. For example, temporal graphs have varying length random walks because walks in temporal graphs have a starting point and an end point. Thus, [16] split a temporal walk into smaller fixed-length truncated walks and connect these walks using time budget, which is defined as the time left before the end time of the temporal walk. So in TG-GAN, the generator generates truncated temporal walks during the training phase and assemble time-budgeted temporal walks in the inference phase. This technique helps to ensure the time validity of temporal graphs.

Chapter 3

Approach

3.1 Applications

The applications we are using are MiniFE, MILC and LAMMPS. MiniFE is an implicit finite element mini-application from Mantevo Project [9]. MILC (the MIMD Lattice Computation [4]) and LAMMPS (the Large-scale Atomic/Molecular Massively Parallel Simulator [13]) are two full applications. MILC is is a Quantum Chromodynamics (QCD) code for SU(3) lattice gauge theory. LAMMPS, which is a crucial simulation workload for the U.S. Department of Energy, is a classical and representative molecular dynamics code provided by Sandia National Laboratories.

3.2 Application Trace

The datasets we are using are MPI (message passing interface) application communication traces of MILC, MiniFE and LAMMPS collected by LogGOPSim [11], which an application simulator based on the popular LogP model [5]. All MILC, MiniFE and LAMMPS, provided by Sandia National Laboratories, are collected from execution runs of 128 parallel processes. All the communication events that took place during an execution instance are recorded, including send, receive, sendreceive, isend, ireceive, allgather, broadcast and allreduce. The first five events are one-to-one communications, while the others are one-to-all or all-to-one communications. A communication event can be represented by a tuple {etype, stime, etime, src, dst, msgsize, type} as described in Table 3.1. Examples are shown in Table 3.2.

Communication Event Description	
Symbol	Description
etype	the type of event
stime	the time the event was initiated
etime	the time the event was completed
src	the set of ids of the sender process
dst	the set of ids of the receiver processes
msgsize	the size of the message
type	the data type of the message

Table 3.1: Descriptions for each component of a communication event

Communication Event Description						
etype	stime	etime	src	dst	msgsize	type
send	1504215489121785	1504215489121789	send process	1	296	1
allgather	1501354915099265	1501354915099289	all processes	all processes	1	4

Table 3.2: Examples of some communication events

3.3 Workflow

Figure 3.1 shows the workflow of our work. We first preprocess data by unifying data format, combining all trace files and splitting and sampling datasets. Then we use adjusted TG-GAN to generate responding synthetic data. Finally we evaluate the performance of our model in three perspectives.

All the experiments are executed on a 16G Tesla P100 GPU standard machine. TG-GAN is publicly available on TG-GAN GitHub repository. Other packages and libraries used are networkx [8] and tensorflow 1.14.0 [1].

3.3.1 Data Preprocessing

In order to unify the data format, we convert one-to-all and all-to-one communications to one-to-one communications. For example, since on LAMMPS

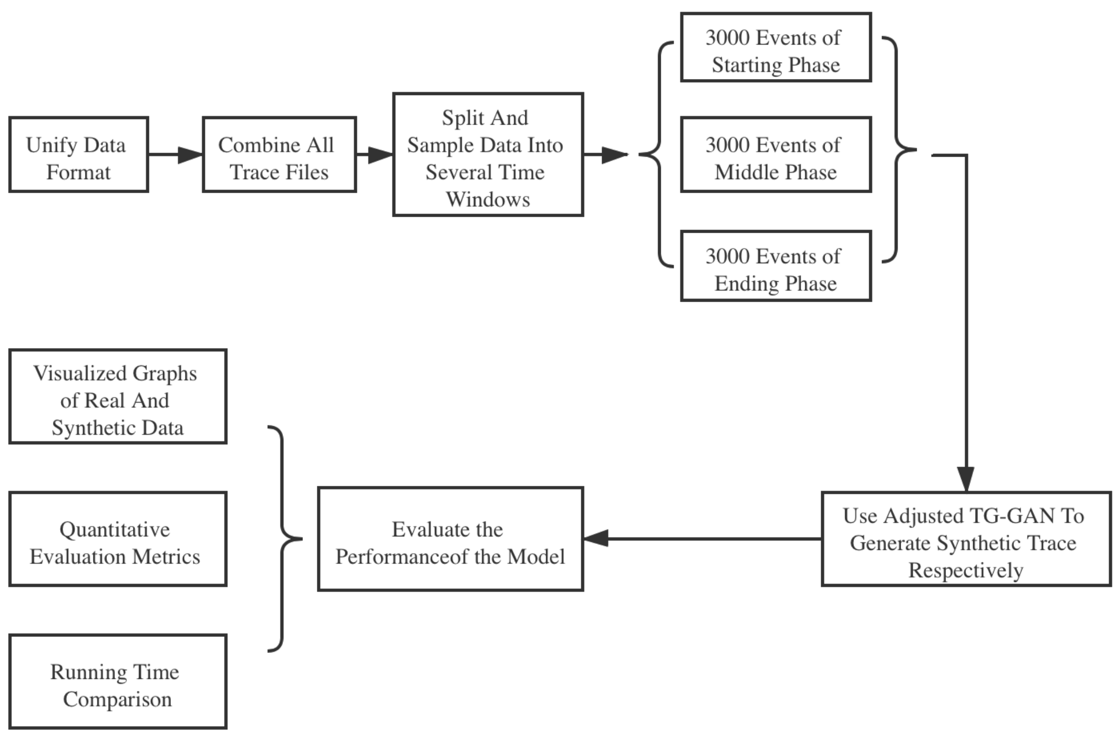

Figure 3.1: The workflow of our work.

dataset allgather event means gathering data from all 128 processes and distributing the combined data to all 128 processes, we convert allgather to 128 receive event from all processes and 128 send event to all processes.

Besides, for each dataset, originally they have separate trace files for every process. However, these trace files have some overlapped events. For example, send event from process one to process two can also be receive event from process two to process one. Therefore, we combine all trace files as one for better representing dataset.

Now, we can consider our trace dataset as a directed graph with edge attributes. Figure 3.2 is visualized from some randomly selected instances from LAMMPS dataset. Each node represents one process, and each edge represents the communicate between two processes. The thickness of the edges represents the communication frequency between two processes. However, the thickness is hard to tell from the graphs because we use logarithm to make our graphs clear since the full graphs have 128 nodes and will be very complex. Therefore, we also utilize heatmaps to illustrate communication frequency between processes clearly. Our purpose is now converted to generate similar synthetic graphs as real graphs.

In order to generate synthetic data that can better represent real data,

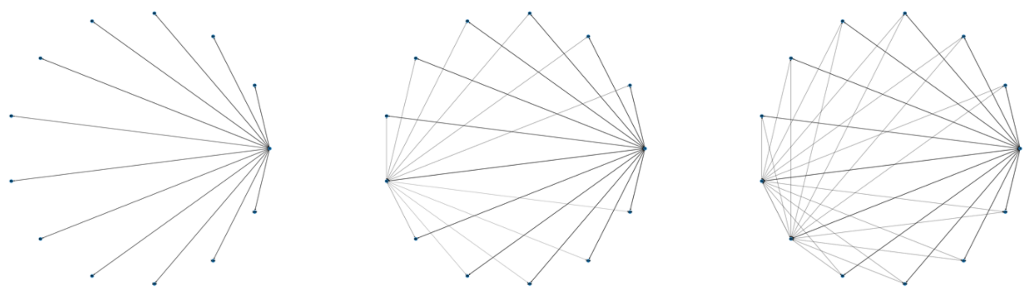

Figure 3.2: Visualizated figures from some randomly selected instances from LAMMPS dataset, thickness are calculated using logrithm methods

we split real data into different time windows because we assume in different communication phases, the application trace will have different patterns. For example, LAMMPS has approximately 5,000,000 instances. So we select three time windows, which are 300,000 to 303,000 instances, 2,200,000 to 2,203,000 instances and 4,580,000 to 4,583,000 instances. After generation, we compare synthetic graphs with real graphs in different time windows.

3.3.2 Adjusted TG-GAN

TG-GAN has a generator using LSTM [10] structure to generate truncated temporal random walks with time stamps, send nodes and receive nodes, a discriminator to detect whether a random walk is real or not, and samplers to extract generated random walks. In order to make TG-GAN more in line with our problem, we adjust the generator as shown in Figure 3.3, where instead of generating [time stamp, send node, receive node] as in TG-GAN, we generate [send time, receive time, send node, receive node, data type, data count].

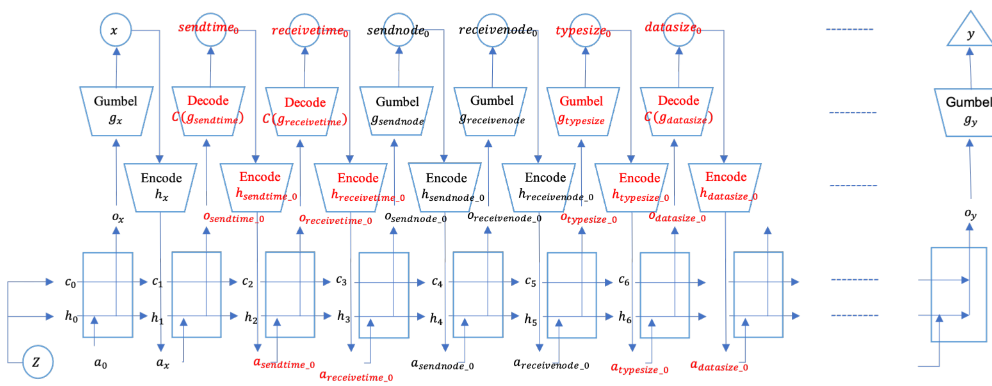

Figure 3.3: The adjusted generator. The adjusted parts are labeled as red.

3.3.3 Evaluation Methods

We evaluate the performance of our method in three perspectives: whether the synthetic data reflect the topology features of the real data, whether the synthetic data reflect the intensity of the real data, and how long does it take to generate synthetic data.

First, we want to evaluate whether the generated traces reflect the communication topology and temporal features of real traces. For one thing, we evaluate the performance using quantitative methods. We calculate closeness centrality, betweenness centrality[6], out degree centrality and in degree centrality using maximum mean discrepancy. Centrality and degree both identify the most important vertices in a graph. So we believe these quantitative methods can describe the topology features of graphs well.

Closeness centrality, defined as

$$C(x) = \frac{1}{\sum_y d(x, y)}, \tag{3.1}$$

where $d(x, y)$ is the distance between node x and y, is a measure of centrality.

Betweenness centrality of a node v is defined as

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{3.2}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v. These two centrality metrics are all based on shortest paths. For the other thing, we compare the visualized network graphs between real data and synthetic data. Through the visualized graphs, we compare the synthetic graphs with the real graphs in same time window to ensure that the generated static features are similar with real data. Also, we evaluate time constraints by comparing the trends of consecutive discrete-time graphs between synthetic graphs and real graphs.

Second, in order to tell whether the synthesized traces reflect the communication intensity of the real traces, we also utilize heatmaps to illustrate communication frequency between processes.

Third, we also compare running time between obtaining real trace and generating synthetic trace of the same number of communication events using our adjusted TG-GAN model.

Chapter 4

Results and Analysis

4.1 Topology and Temporal Features

To compare the topology features of real data and synthetic data, we evaluate model performance using quantitative methods as shown in Table 4.1.

For the synthetic trace generation experiments, we do with nine traces mapping to different phases from three applications. The closeness centrality ranges from 0.0037 to 0.2000, with 0.0975 mean and 0.0846 std. The betweeness centrality ranges from 0.0000 to 0.1387, with 0.0335 mean and 0.0433 std. The out degree ranges from 0.0003 to 0.1989, with 0.0810 mean and 0.0858 std. The in degree ranges from 0.0002 to 0.1992, with 0.0798 mean and 0.0850 std.

All these four evaluation metrics describe graphs in node level and are all standardized. These evaluation metrics have good results in most of the datasets. Therefore, the quantitative results in Table 4.1 show the topology similarity between synthetic graphs and real graphs.

Distances Between Real And Synthetic Graphs Using MMD				
dataset	closeness centrality	betweenness centrality	out degree	in degree
LAMMPS phase 1	0.0792	0.0000	0.0792	0.0792
LAMMPS phase 2	0.0054	0.0000	0.0003	0.0002
LAMMPS phase 3	0.1546	0.0271	0.0256	0.0245
MINIFE phase 1	0.1979	0.0651	0.1989	0.1992
MINIFE phase 2	0.2000	0.0491	0.1985	0.1990
MINIFE phase 3	0.0170	0.0020	0.0032	0.0033
MILC phase 1	0.2000	0.1387	0.1967	0.1891
MILC phase 2	0.0201	0.0197	0.0245	0.0221
MILC phase 3	0.0037	0.0000	0.0019	0.0020

Table 4.1: Quantitative evaluation for synthetic data

Also, we visualize network graphs of both real data and synthetic data to illustrate the topology features and temporal features. Figure 4.1 is the visualization of middle phase of LAMMPS dataset. The upper left graph is the first 1000 instances, the upper middle is the first 2000 instances and the upper right is all 3000 instances of the selected time window in real data. And graphs in the second and third row are corresponding synthetic graphs with different sampler parameters. Figure 4.2 is the visualization of middle phase of MILC dataset. Figure 4.3 is the visualization of middle phase of

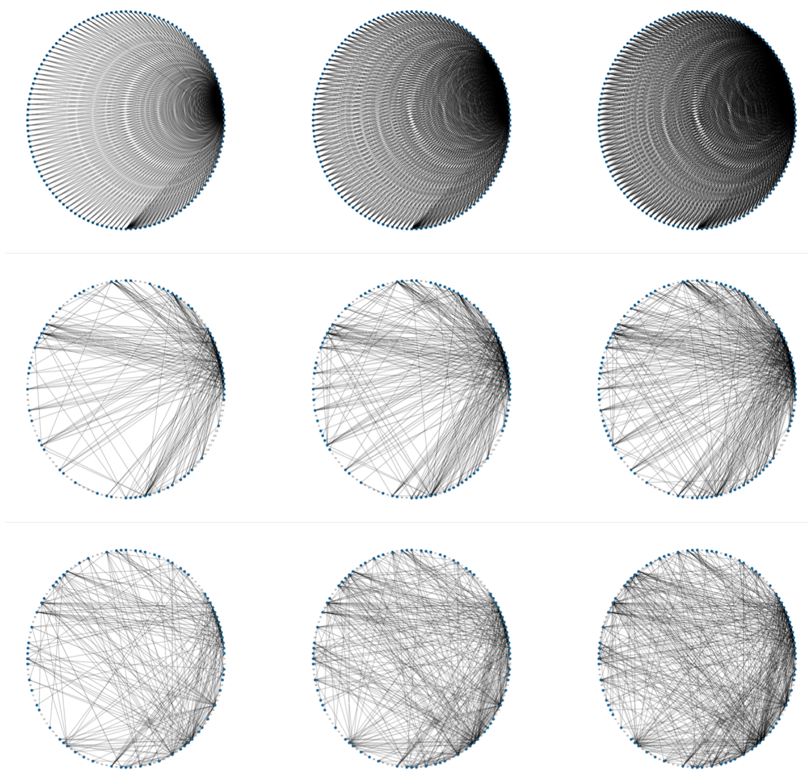

Figure 4.1: First row: real data in LAMMPS, from 2200000 to 2203000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2;

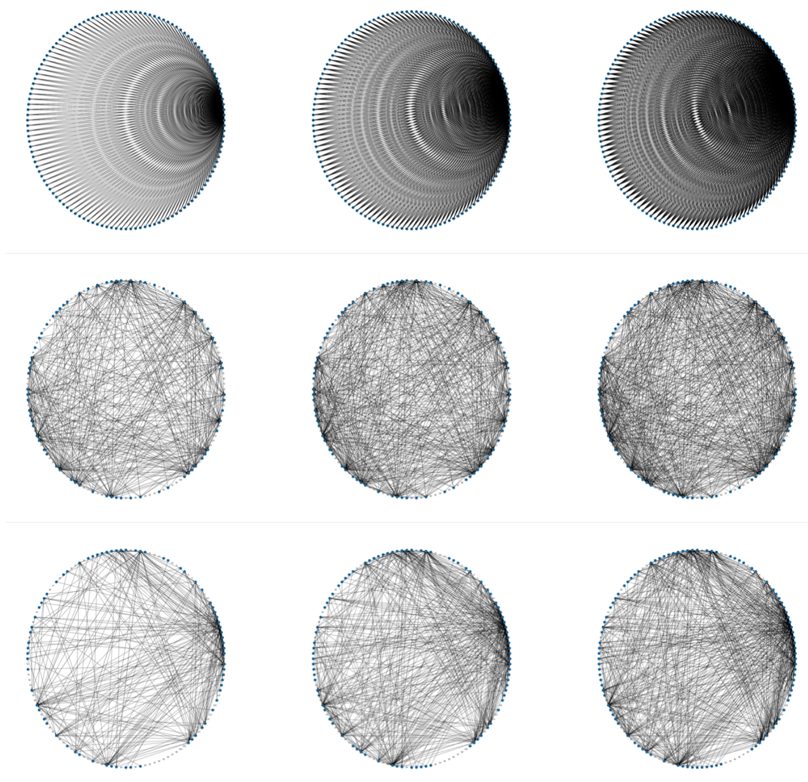

Figure 4.2: First row: real data in MILC, from 18000000 to 18003000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2;

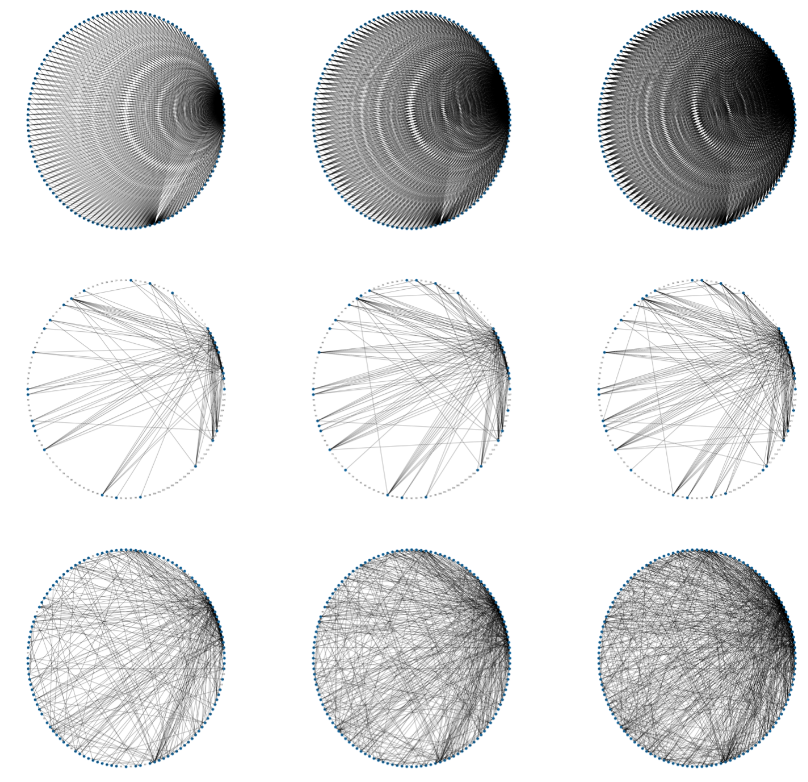

Figure 4.3: First row: real data in MINIFE, from 6000000 to 6003000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2;

MINIFE dataset. The complete results are in the appendix A.

From the visualized network graphs, we find that the synthetic graph have very similar concentrated nodes as the real graph, especially for middle and ending phase. For example, in Figure 4.3, the nodes with a greater out degree are concentrated on the upper right and right below for both synthetic graphs and real graphs.

Besides, the trends of consecutive discrete-time graphs between synthetic graphs and real graphs are similar. For example, in Figure 4.1, the degree of nodes in up right increases for both synthetic graphs and real graphs, showing that the temporal validity has been maintained in synthetic graphs.

The visualized network graphs show that TG-GAN has the potential ability to generate synthetic data with similar topology features and temporal features converging to real data, but more research needs to be done to yield further refined results.

4.2 Communication Intensity

Since it is hard to directly see communication network intensity from visualized network graphs, we also utilize the comparison of heatmaps between real and synthetic graphs. Figure 4.4 is the heatmaps of middle phase of LAMMPS dataset. The left one is the heatmap of real data and the middle and right ones are the heatmaps of synthetic data with different sampler parameters. Figure 4.5 is the heatmaps of middle phase of MILC dataset, and Figure 4.6 is the heatmaps of middle phase of MINIFE dataset. The complete results are in the appendix A.

From the heatmaps, we can see that the synthetic graphs are sparser and have less communication intensity than the real graphs. Although the sampler parameter can help fine-tune the density of the generated graphs, the generated graphs are still very different from real graphs and there is a need for improvement in this area.

It is a limitation when we want to evaluate system performance through entire application traces, because full application traces as the real traces are required for system performance assessment. However, in some cases, a small window of trace is enough for evaluating specific aspects of system performance.

We can also see some similar patterns between synthetic and real data from heatmaps. For example, in Figure 4.4, the middle figure shows the similar concentration in upper left as in the left figure, which means the corresponding synthetic data has similar patterns as the real data.

Therefore, the heatmaps show that TG-GAN may have the ability to generate high-quality synthetic graphs with similar communication intensity as real graphs, and further exploration is needed.

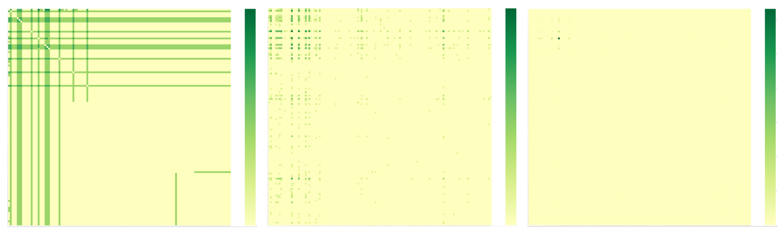

Figure 4.4: First: heatmap of real data in LAMMPS, from 2200000 to 2203000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2;

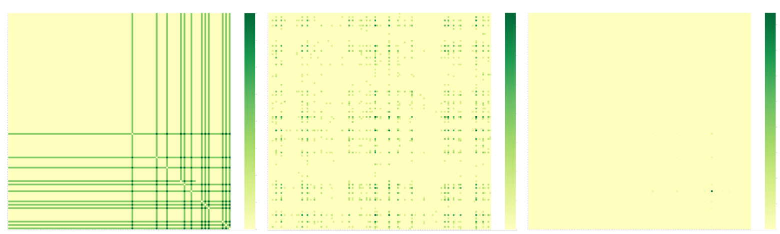

Figure 4.5: First: heatmap of real data in MILC, from 18000000 to 18003000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2;

4.3 Synthetic Generation Time

Besides, we compare the running time for obtaining real trace and synthetic trace as shown in Table 4.2. The running time is represented by CPU

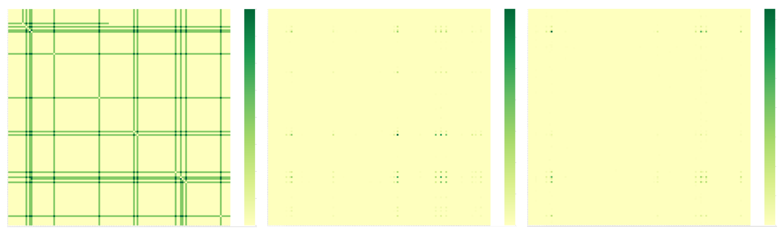

Figure 4.6: First: heatmap of real data in MINIFE, from 6000000 to 6003000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2;

time, which means $wallclock * number\ of\ processors$. The running time for generating synthetic trace is calculated by

$$time = \frac{1}{3} \sum_{i=1,2,3} \frac{time_i * n_i}{m_i}, \tag{4.1}$$

where i represents three different time window, $time_i$ is running time for each time window, n_i is the number of communication events in real trace and m_i is the number of generated synthetic communication events.

The running time for generating synthetic trace is significantly more than collecting real trace as shown in Table 4.2. However, for one thing, these results were generated on a computer with very basic computational power. An avenue of exploration is to see if TG-GAN can be optimized for better concurrency to speed up processing times. For another thing, according to [16], TG-GAN has a constant running time in terms of number of nodes, which means generating synthetic trace using TG-GAN will have an advantage over collecting real trace when the number of processes increases.

Thence, using TG-GAN to generate synthetic trace will help save time and resources when the number of processes is large.

Running Time Comparison		
dataset	real trace (s)	syntetic trace (s)
LAMMPS	13,322	71,246
MINIFE	22,189	33,863
MILC	3,940	18,565

Table 4.2: Running time comparison for obtaining real trace and generating synthetic trace

Chapter 5

Summary and Future Work

5.1 Summary

In summary, we studied a machine learning based synthetic data generation method that adjusts TG-GAN to generate synthetic application trace, which can be represented by temporal directed graphs with edge attributes. We evaluated the performance of synthetic data using quantitative metrics, visualized network graphs, heatmaps and running time. Our quantitative and visualization results show that while our current approach has the potential to generate synthetic traces that maintain the topological and temporal features of the real traces, it does not do a good job for maintaining communication intensity features. Besides, generating synthetic data has an advantage over collecting real data when the number of processes increase. In general, the model is efficient and able to generate synthetic data with similar patterns as the real data. We believe the adjusted TG-GAN has very good potential for generating high-quality synthetic application trace and help save time and resources.

5.2 Future Work

While our TG-GAN approach shows a lot of promise, we can take more aspects into account to improve model performance and further enrich the experiments.

First, we can do more experiments to tune parameters. Second, we can utilize more methods for evaluation. For example, we can run synthetic data through simulator like LogGOPSim [11] to see the performance compared with real data to evaluate performance from HPC perspective. Third, to further enrich our experiments, we can also compare TG-GAN with other graph generation methods, like NetGAN, GraphVAE, GraphRNN and TagGen. Last, in our evaluation phase, we consider continuous-time graphs as several snapshots to evaluate the temporal constraint features of the synthetic data. However, we can also directly evaluate continuous-time graphs by reporting the mean of specific graph measures, such as mean degree, based on a set of graph samples.

Appendix A

The Complete Set of Visualization Results

Here we present all of the network graph and heatmap visualizations for LAMMPS, MILC and MINIFE synthetic data generated as a part of this study.

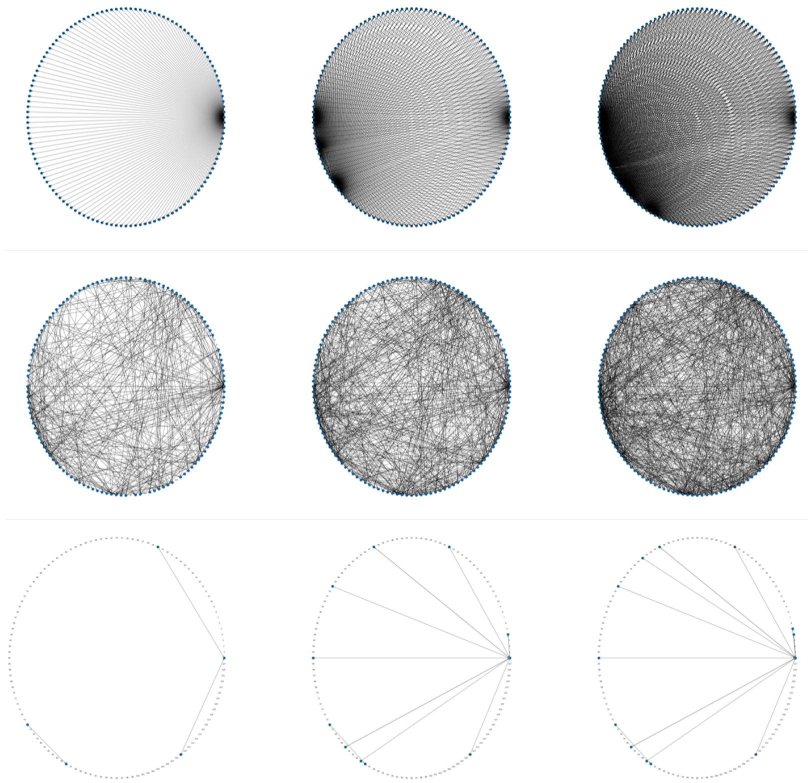

Figure A.1: First row: real data in LAMMPS, from 300000 to 303000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2;

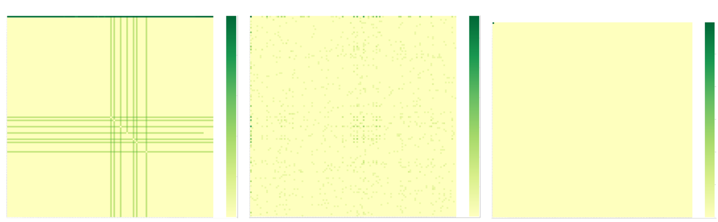

Figure A.2: First: heatmap of real data in LAMMPS, from 300000 to 303000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2;

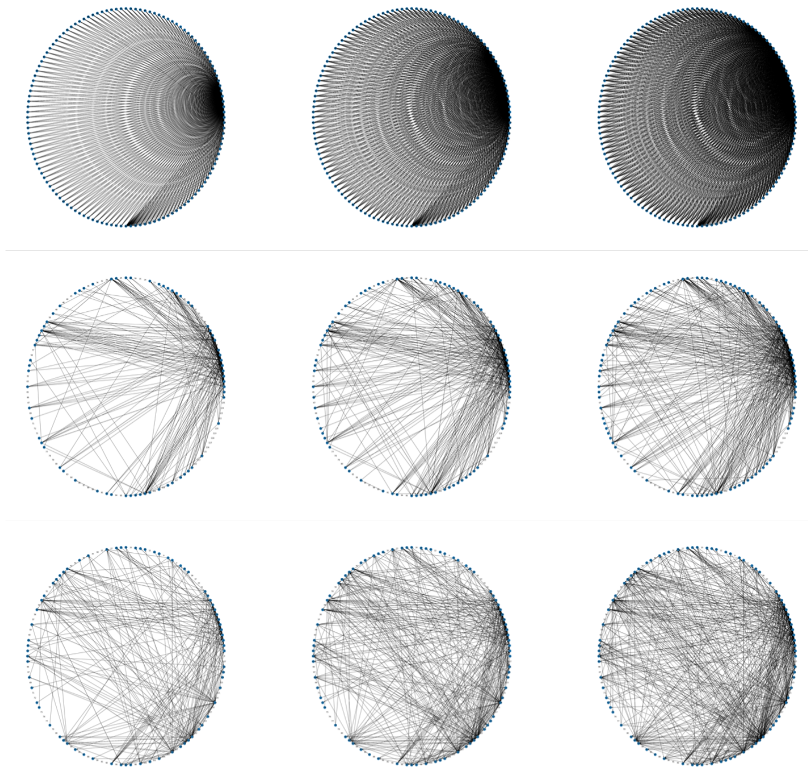

Figure A.3: First row: real data in LAMMPS, from 2200000 to 2203000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2;

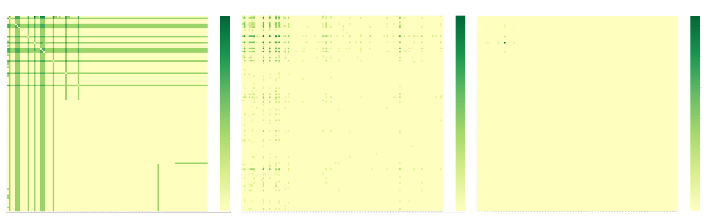

Figure A.4: First: heatmap of real data in LAMMPS, from 2200000 to 2203000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2;

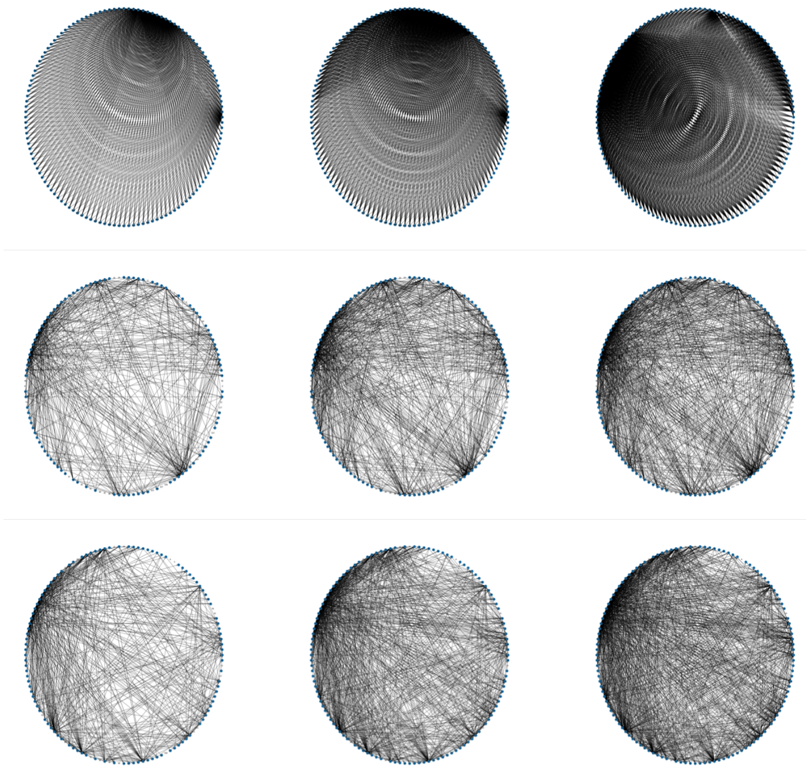

Figure A.5: First row: real data in LAMMPS, from 4580000 to 4583000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2;

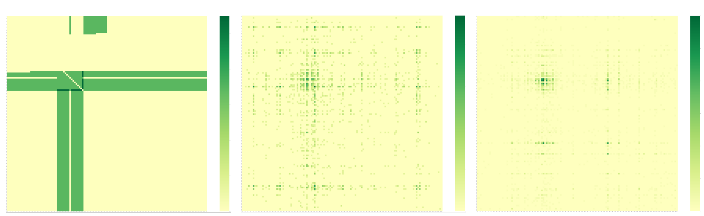

Figure A.6: First: heatmap of real data in LAMMPS, from 4580000 to 4583000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2;

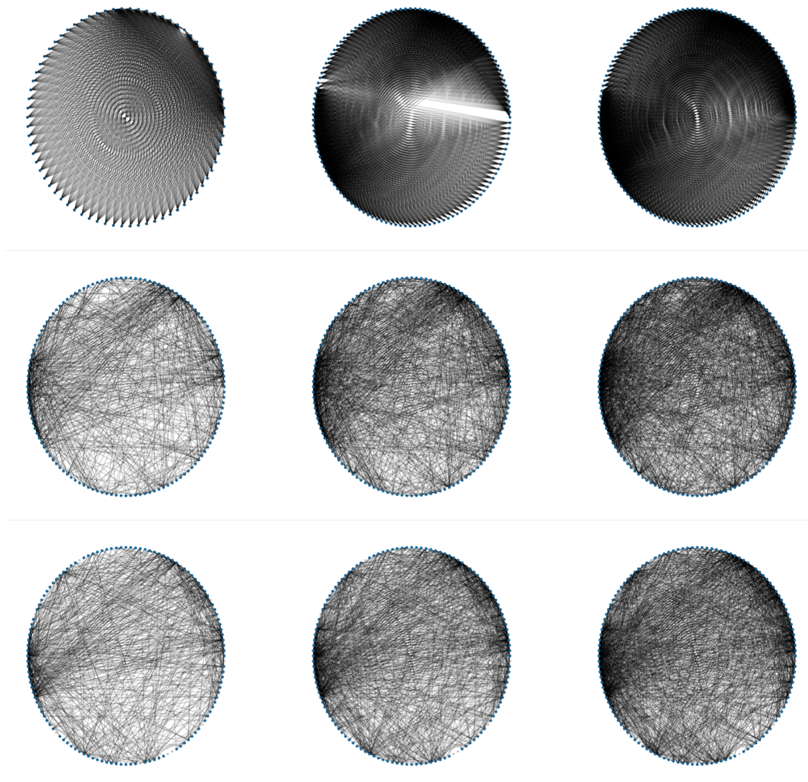

Figure A.7: First row: real data in MILC, from 300000 to 303000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2;

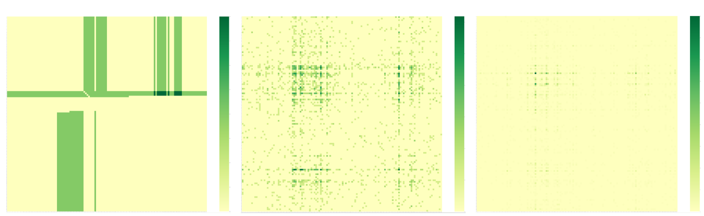

Figure A.8: First: heatmap of real data in MILC, from 300000 to 303000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2;

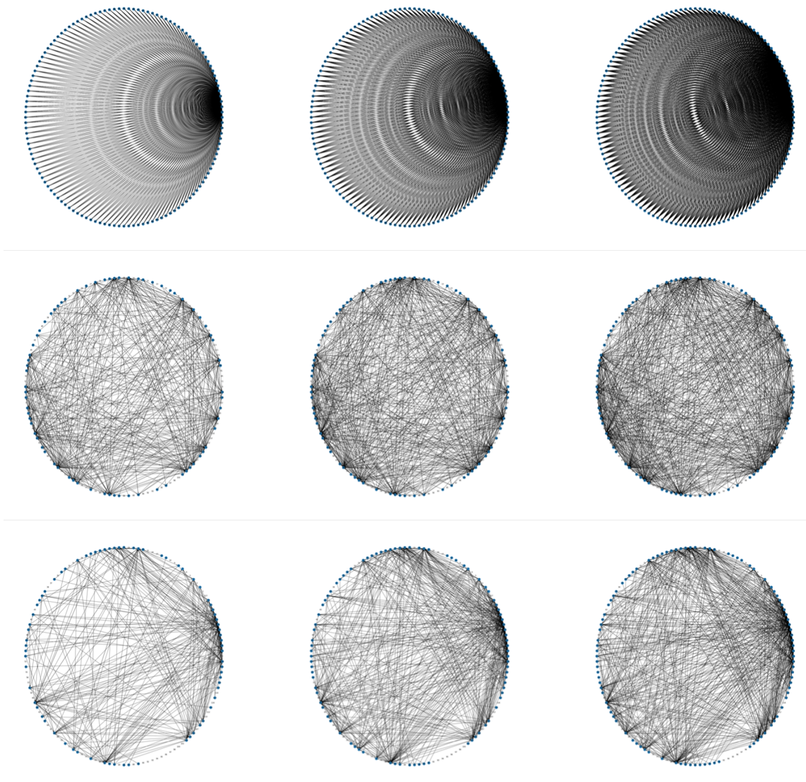

Figure A.9: First row: real data in MILC, from 18000000 to 18003000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2;

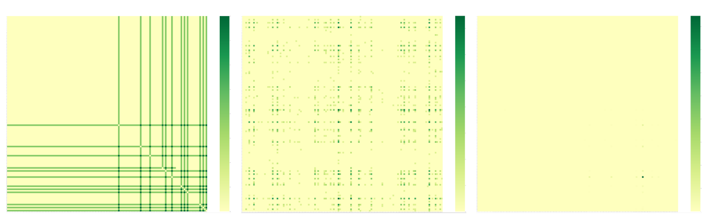

Figure A.10: First: heatmap of real data in MILC, from 18000000 to 18003000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2;

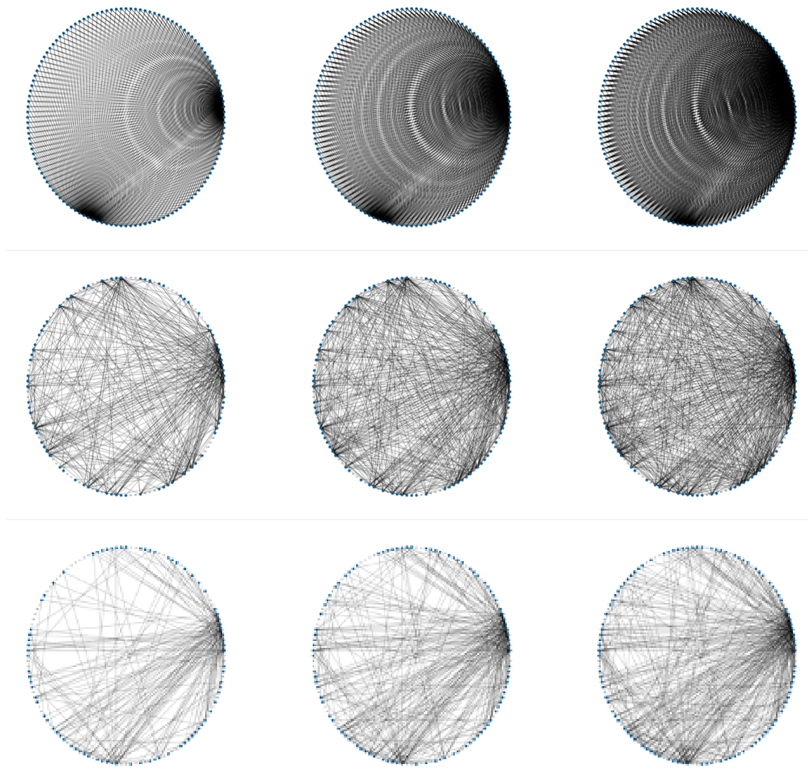

Figure A.11: First row: real data in MILC, from 37000000 to 37003000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2;

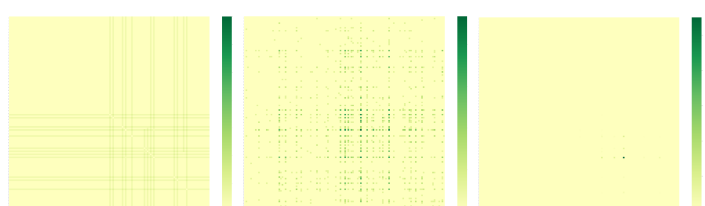

Figure A.12: First: heatmap of real data in MILC, from 37000000 to 37003000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2;

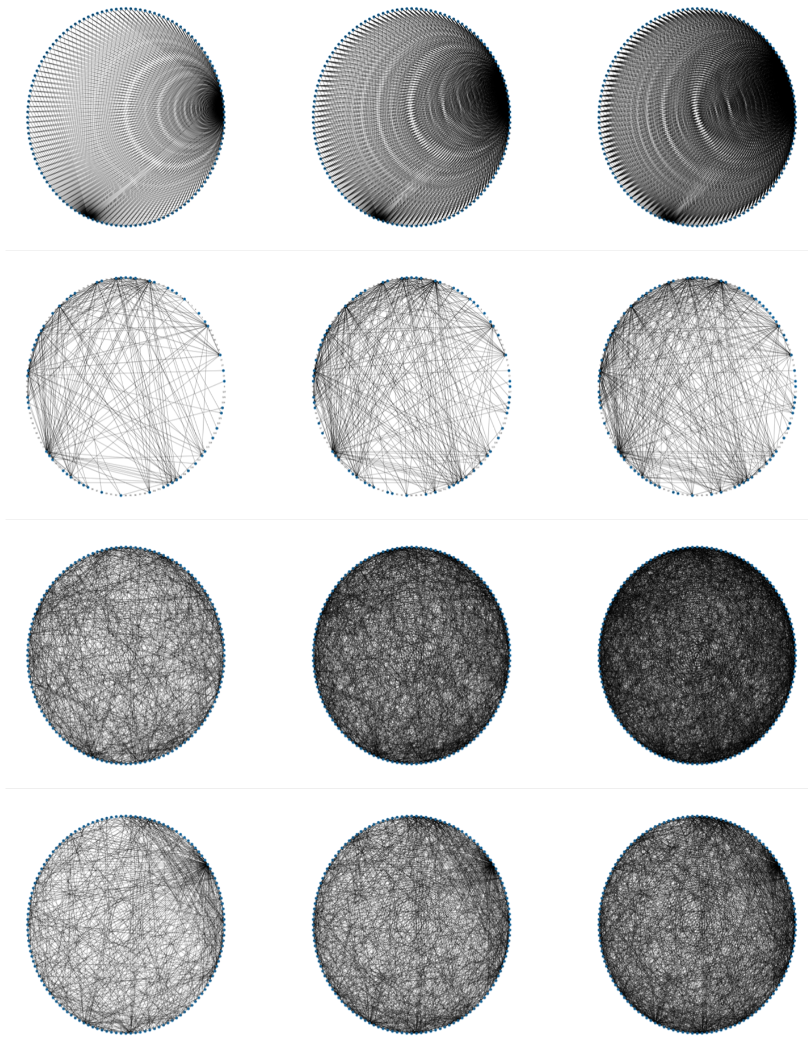

Figure A.13: First row: real data in MINIFE, from 300000 to 303000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2; Forth row: corresponding synthetic data, sampler parameter is 0.05

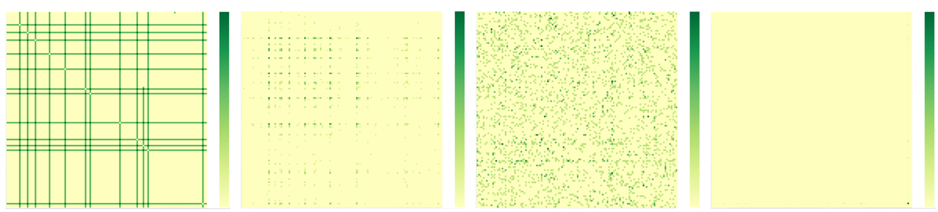

Figure A.14: First: heatmap of real data in MINIFE, from 300000 to 303000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2; Forth: corresponding synthetic data, sampler parameter is 0.05

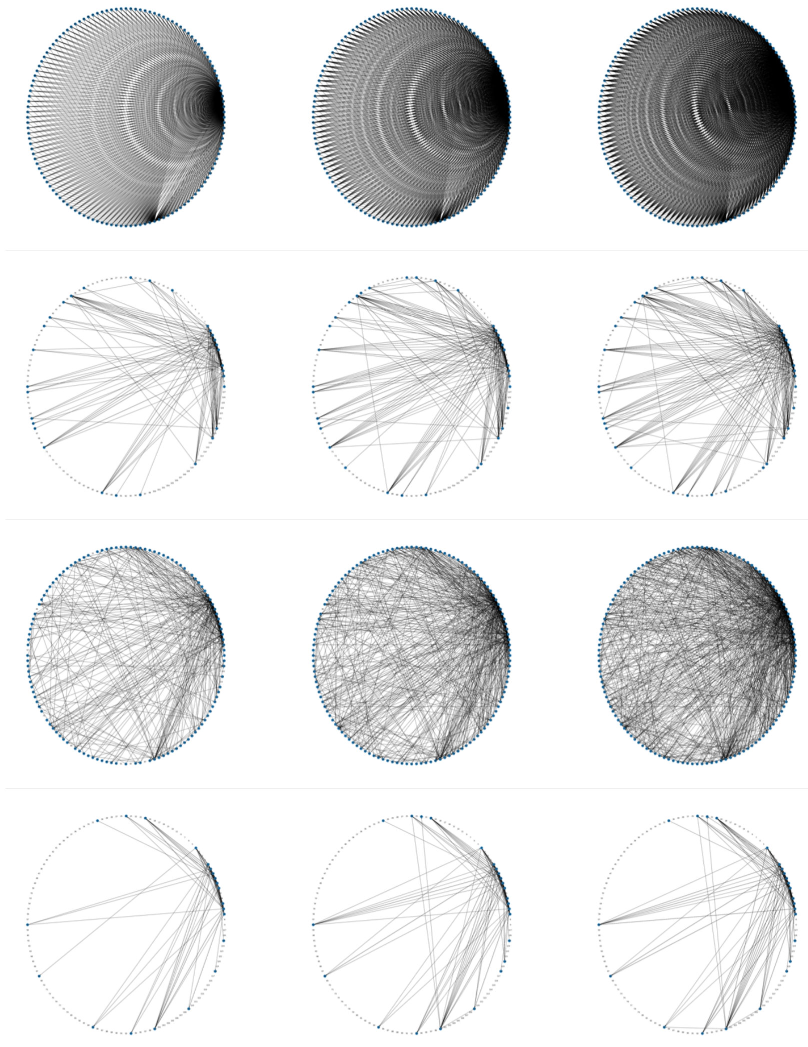

Figure A.15: First row: real data in MINIFE, from 6000000 to 6003000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2; Forth row: corresponding synthetic data, sampler parameter is 0.05

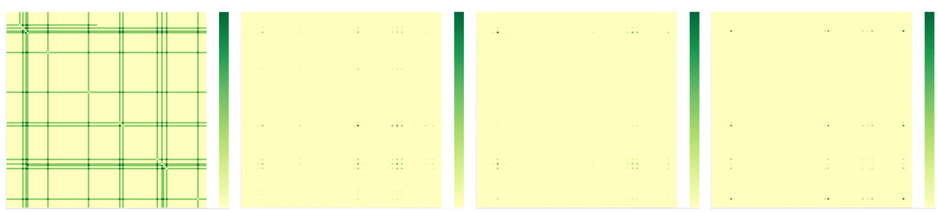

Figure A.16: First: heatmap of real data in MINIFE, from 6000000 to 6003000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2; Forth: corresponding synthetic data, sampler parameter is 0.05

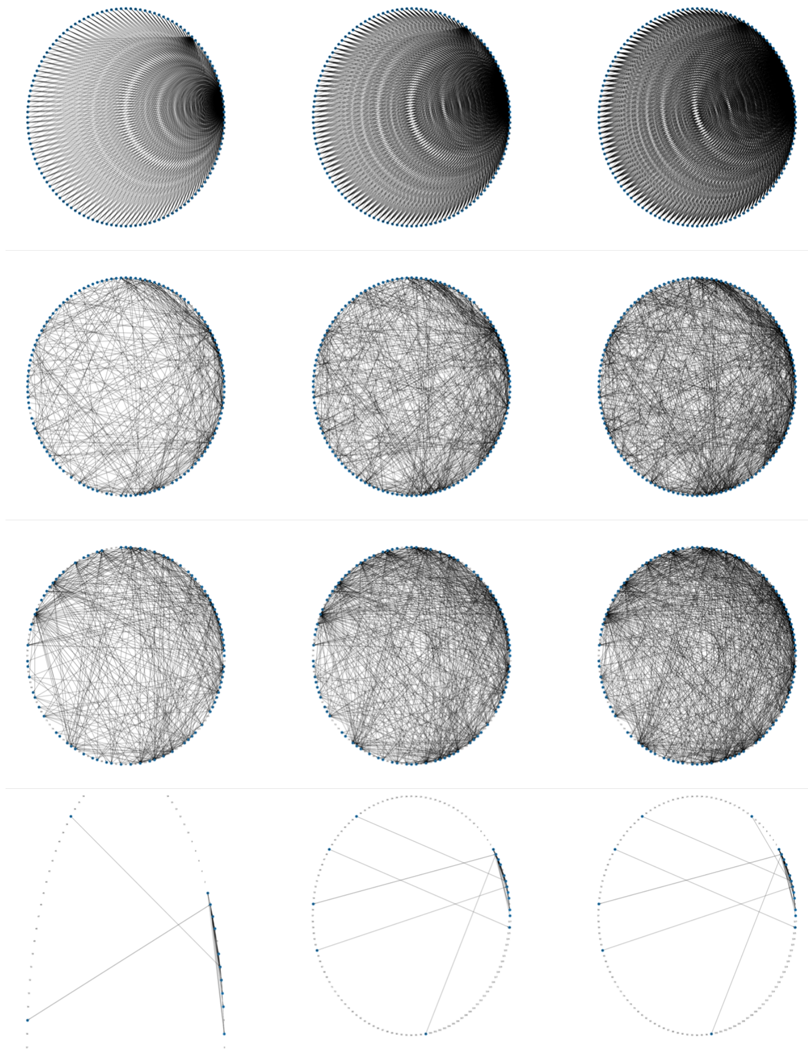

Figure A.17: First row: real data in MINIFE, from 14000000 to 14003000 instances; Second row: corresponding synthetic data, sampler parameter is 0.5; Third row: corresponding synthetic data, sampler parameter is 0.2; Forth row: corresponding synthetic data, sampler parameter is 0.05

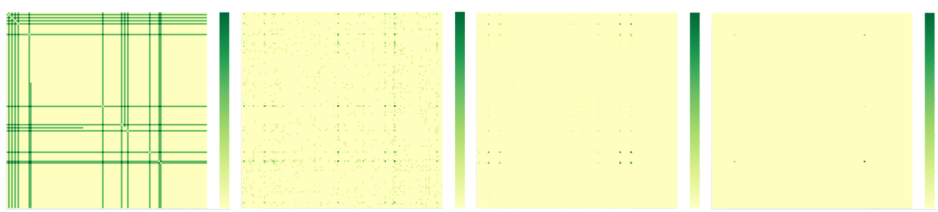

Figure A.18: First: heatmap of real data in MINIFE, from 14000000 to 14003000 instances; Second: heatmap of the corresponding synthetic data, sampler parameter is 0.5; Third: corresponding synthetic data, sampler parameter is 0.2; Forth: corresponding synthetic data, sampler parameter is 0.05

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[2] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. NetGAN: Generating graphs via random walks. In Jennifer

Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 610–619. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/bojchevski18a.html.

[3] Edward Choi, Siddharth Biswal, Bradley A. Malin, Jon Duke, Walter F. Stewart, and Jimeng Sun. Generating multi-label discrete electronic health records using generative adversarial networks. *CoRR*, abs/1703.06490, 2017. URL http://arxiv.org/abs/1703.06490.

[4] MILC Collaboration. Mimd lattice computation (milc) collaboration home page. Information available at http://physics.indiana.edu/sg/milc.html.

[5] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten Von Eicken. Logp: Towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 1–12, 1993.

[6] Linton C. Freeman. A set of measures of centrality based on betweenness.

Sociometry, 40(1):35–41, 1977. ISSN 00380431. URL `http://www.jstor.org/stable/3033543`.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27, pages 2672–2680. Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf`.

[8] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[9] Michael A Heroux, Douglas W Doerfler, Paul S Crozier, James M Willenbring, H Carter Edwards, Alan Williams, Mahesh Rajan, Eric R Keiter, Heidi K Thornquist, and Robert W Numrich. Improving perfor-

mance via mini-applications. *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, 3, 2009.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[11] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Loggopsim: Simulating large-scale applications in the loggops model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, page 597–604, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589428. doi: 10.1145/1851476.1851564. URL `https://doi.org/10.1145/1851476.1851564`.

[12] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[13] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995. ISSN 0021-9991. doi: https://doi.org/10.1006/jcph.1995.1039. URL `https://www.sciencedirect.com/science/article/pii/S002199918571039X`.

[14] Martin Simonovsky and Nikos Komodakis. GraphVAE: Towards gen-

eration of small graphs using variational autoencoders, 2018. URL `https://openreview.net/forum?id=SJlhPMWAW`.

[15] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: A deep generative model for graphs. *CoRR*, abs/1802.08773, 2018. URL `http://arxiv.org/abs/1802.08773`.

[16] Liming Zhang, Liang Zhao, Shan Qin, and Dieter Pfoser. Tg-gan: Continuous-time temporal graph generation with deep generative models, 2020.

[17] Dawei Zhou, Lecheng Zheng, Jiawei Han, and Jingrui He. A data-driven graph generative model for temporal interaction networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '20, page 401–411, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403082. URL `https://doi.org/10.1145/3394486.3403082`.