Designing GNN Training Data with Limited Samples and Small Network Sizes

snowyowl Team
Junior Momo Ziazet[1], Charles Boudreau[1], Brigitte Jaumard[1], Oscar Delgado[1]
[1] *Computer Science and Software Engineering, Concordia University, Montreal, Canada*

November 28, 2022

1 Introduction

The quality of machine learning based models is determined by two major factors: the model architecture and the training dataset. Given a Graph Neural Network (GNN) model, the goal of this work is to produce a training dataset of limited size (at most 100 samples with networks of at most 10 nodes) that should help the given GNN model to scale effectively to samples of larger networks (50 to 300 nodes) than those seen during training.

The process we employed to generate our dataset can be divided into the following 3 steps:

1. **Initial dataset generation:** We try to match the distribution of the validation dataset for parameters that do not depend on the size of the network
2. **Refactoring the dataset:** In order to add some variability to the dataset to account for different use cases.
3. **Dataset cleaning:** In order to keep only high quality data rather than focusing on the volume of data.

The following sections 2, 3 and 4 will describe each of these steps in detail.

2 Initial dataset generation: Matching the validation set

As the goal of the project is to build a dataset that will allow a model trained on it to obtain good prediction results on data possessing a similar distribution to the data of the validation set, we began by conducting a detailed analysis of the validation set in order to extract information that will allow us to make some assumptions on how some parameters of the validation sets have been generated. For example, we wanted to elaborate a hypothesis on how the traffic and the type of service is assigned to the source and destination node pairs, what is the average length of a path, what are the values of the capacities of the links and how these capacities are assigned to the links, what are the buffer sizes of the nodes, how are they assigned and what scheduling policies are used, etc. Tables 1 and 2 show some of the parameters, found by our observation of the validation set. We can identify certain parameters that do not vary as the size of the graph changes within the validation set, and we assumed at this stage that we should also set these parameters in our training set. We then attempted to generate a dataset which had similar characteristics while still complying with the restrictions put forth by the organizers. Setting the parameters in Tables 1 and 2 resulted in a MAPE around 27-28 %.

3 Refactoring the dataset

Trying to match the parameters of the validation set with networks of at most 10 nodes and with only 100 samples did not allow us to obtain acceptable performance. We interpreted this as our dataset not being diverse enough and not accounting for the appropriate amount of variability in the parameters that can help accommodate the full range of possible use cases or scenarios encountered during testing. We then move forward to analyse other parameters that may affect the parameters we set in Section 2, such as the ranges for delay, queue utilization, link utilization and avg port occupancy, in order to understand what was missing.

Table 1: Hypothesis made on graph nodes and edges after validation set analysis

Parameters	Values	Probabilities
Policies	[FIFO, SP, WFQ, DRR]	[0.25, 0.25, 0.25, 0.25]
Buffer Sizes	[8000, 16000, 32000, 64000]	[0.25, 0.25, 0.25, 0.25]
WFQ weights	["70,20,10","33.3, 33.3, 33.4","60,30,10","80,10,10","65,25,10"]	[0.2, 0.2, 0.2, 0.2, 0.2]
DRR weights	["60,30,10","70,25,5","33.3,33.3,33.4","50,40,10","90,5,5"]	[0.2, 0.2, 0.2, 0.2, 0.2]
Link capacity	[10000, 25000, 40000, 100000, 250000, 400000, 1000000]	based on number of nodes

Table 2: Hypothesis made on flows after validation set analysis

Parameters	Values	Probabilities
Traffic flow	Uniform(0,1)×I, I∈[1000,2000,3000,4000]	[0.25, 0.25, 0.25, 0.25]
Packet Size Distribution	"0,500,0.22,750,0.05,1000,0.06,1250,0.62,1500,0.05"	0.2
	"0,500,0.08,750,0.16,1000,0.35,1250,0.21,1500,0.2"	0.2
	"0,500,0.53,750,0.16,1000,0.07,1250,0.1,1500,0.14"	0.2
	"0,500,0.1,750,0.16,1000,0.036,7,1250,0.24,1500,0.14"	0.2
	"0,500,0.05,750,0.28,1000,0.25,1250,0.27,1500,0.15"	0.2
Time Distribution	"Poisson","CBR","ON-OFF (5,5)"	[1/3, 1/3, 1/3]
Type of Service	0,1,2	[0.1, 0.3, 0.6]

3.1 Link capacity assignment

Our exploration of the validation dataset showed that the average bandwidth of the links in a given topology generally increased proportionally to the number of nodes. We suppose this is done to accommodate the increasing number of flows in larger topologies, as the validation abides by the same rules as the training set in that each node pair has one flow assigned to it. One restriction put forth by the organizers of the competition is that, in our dataset, we can only have one flow per node pair. When taken together with the restriction that training set topologies must not exceed 10 nodes, this means that the total number of flows will always be less than the amount found in the validation set. Indeed, the link utilization analysis we made showed that our link utilization was very low on average (around 40%). In order to adjust the link utilization and solve this issue, we designed two strategies:

- **Strategy 1: Link Capacity based on Routing:** In this strategy, the routing policy is known in advance and we set the capacity of the links to be proportional to the traffic it encounters. We can therefore set the capacity so that we get a desired utilization. Equation 1 shows how we obtained the link capacities.

$$\text{Capacity}_l = \frac{\sum\limits_{f \in N_\ell} t_f}{\text{link\_utilization}_l}, \qquad (1)$$

where link_utilization is chosen from a normal distribution of mean μ in the set $S = \{0.2, 0.4, 0.6, 0.8\}$ and standard deviation $\sigma = \frac{\mu}{2}$. The different values of the set S have been chosen in order to add some variability to the dataset, every single example yielding a meaningful contribution. Algorithm 1 details the process for obtaining the link utilization. t_f is the traffic of the flow f and N_l is the number of flows traversing link l.

- **Strategy 2: Routing based on Link Capacity:** In this strategy, the link capacity is known in advance and the routing is made such that utilization of a link doesn't exceed a given utilization threshold. This strategy was not used for the final solution, so we won't explain it in detail in this report.

Using strategy 1, the values of the link capacity were concentrated around 10000, which is very small compared to link capacities in the validation set. Looking at how the RouteNet_Fermi model [1] estimates the delay of a flow in Equation 2,

$$\text{delay}_f = \sum_{(q,l) \in f} \left(\frac{gnn\_qo_l}{capacity_l} + \frac{\mu\_fps_f}{capacity_l} \right), \qquad (2)$$

we see that the delay is highly dependent on the link capacity as it is used to calculate the queuing and transmission delay. Since the GNN readout output (gnn_qo_l) may give similar values in the training and validation set, having a link capacity value in the training set that is very different from those in the validation set may be counterproductive. In order to accommodate this, instead of using the link capacity directly deriving from equation 1 (hereinafter referred to as 'cap'), we set the link capacity to be the nearest candidate to 'cap' from the set {10000, 25000, 40000, 100000, 250000, 400000}, which is a subset of the link capacity in the validation set. Algorithm 2 details the process for setting the link capacities in the graphs according to the traffic information.

Algorithm 1: get_load()

```
mean = $[0.2, 0.4, 0.6, 0.8]$
weights = $[0.3, 0.3, 0.3, 0.1]$
while True do
    $\mu$ = random(mean, weights)
    $\sigma = \mu/2$
    link_load = $\mathcal{N}(\mu, \sigma)$
    if 0 <= link_load <= 1 then
        break
Return: link_load
```

Algorithm 2: set_link_bandwidth()

```
Input: G, paths, traffic, capacity_set
link_bw = Array of 0
foreach pair (src, dst) in G do
    path = paths(src,dst)
    foreach e in path do
        link_bw(e(src), e(dts)) += traffic(src,dst)
foreach e in G.edges do
    link_load = get_load()
    cap = link_bw(e(src), e(dst)) / link_load
    $Id\_cap = argmin(|capacity\_set - cap|)$
    $G(e[src], e[dst]) = capacity\_set[Id\_cap]$
Return: G
```

3.2 Network topology choice

Since the maximum number of nodes per network topology was set to 10, we decided to generate only 10-node graphs to have the highest possible number of flows per sample. From our experiments, we found that having one unique network topology per sample, so 100 different network topologies in total, was not necessarily a good idea, as it could add too much variability or noise to the data set. Therefore, we carefully designed and selected 10 network topologies to have two characteristics found in the validation set: the presence of nodes of degree 1 and different node degree configurations as presented in Figure 1 to increase the probability of having a different path length distribution using the shortest path algorithm. Each topology was used to generate 10 samples where only the node and edge characteristics are modified. The node attributes are chosen as described in Table 1 and the edge characteristics are chosen according to Section 3.1.

3.3 Flows generation

Based on our observations of the validation set, we identify four traffic flow intensities: 1000, 2000, 3000, and 4000 bps, which we used as a base. For each sample, we randomly choose an intensity and the flows in that sample will be selected from the interval [intensity/2, intensity]. Another important aspect is that, since we chose to work with only 10 network topologies of 10 nodes each, we found that defining only 10 different traffic matrices gives better performance than 100 different traffic matrices. Thus, we set 10 different traffic matrices per network topology and the same traffic matrices are assigned to the different network topologies. In addition, to match the flow value distribution found in the validation set, the probability distribution of the packet size distribution and time were updated from [0.2, 0.2, 0.2, 0.2, 0.2] to [0.1, 0.2, 0.3, 0.3, 0.1] and from [1/3, 1/3, 1/3] to [0.8, 0.1, 0.1] respectively. Figure 2 presents the traffic flow distribution in the training and validation set.

The different changes described in subsections 3.1, 3.2 and 3.3 resulted in a decrease in the MAPE obtained by the model from 27-28% to 8-10%.

4 Dataset cleaning

At this point, we have 100 samples and are exploring whether trying to clean up this dataset by removing a few samples that may be considered too noisy can help improve the results. The meaning of "noisy" in our case was difficult to specify. Knowing that we were using the shortest path routing algorithm and that the message passing scheme needed 8 iterations, in order to potentially reduce the oversmoothing effect during the training, we decided to remove all samples that did not contain at least one path of length 4 which represents approximately the average path length in the validation set. We identified 10 samples and found that all these samples were generated with the topology containing a degree 6 node. After removing these samples, the MAPE dropped to 6-7% over the entire validation set. Table 3 shows the MAPE obtained for each validation set.

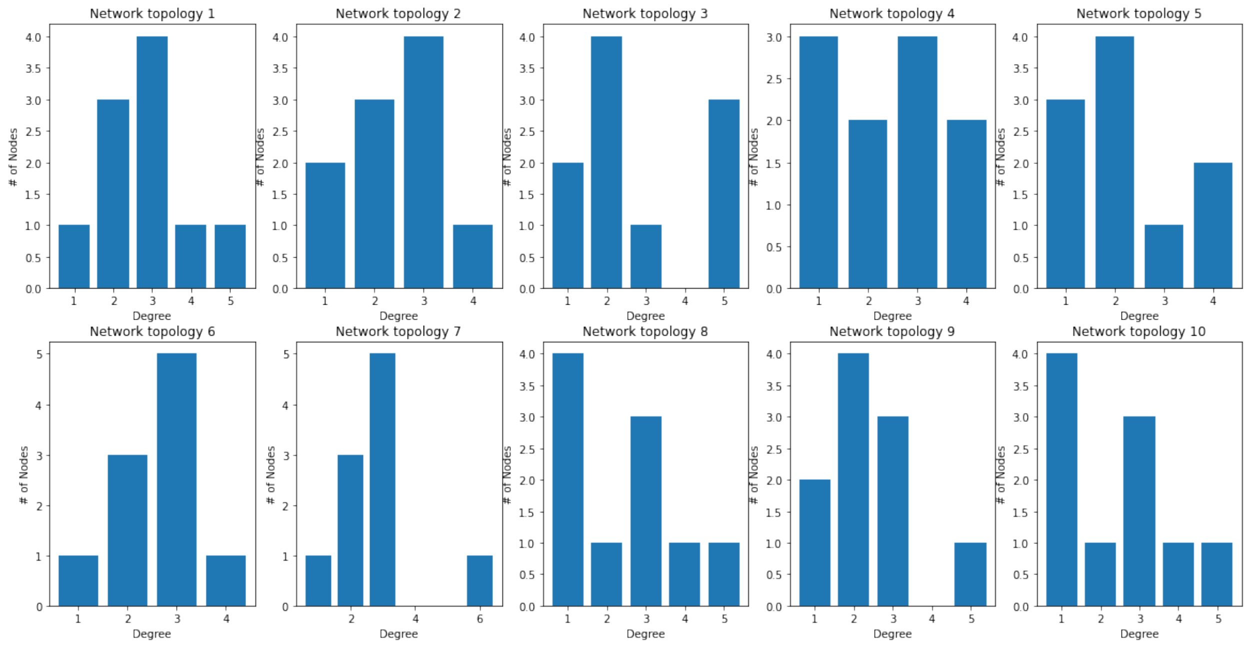
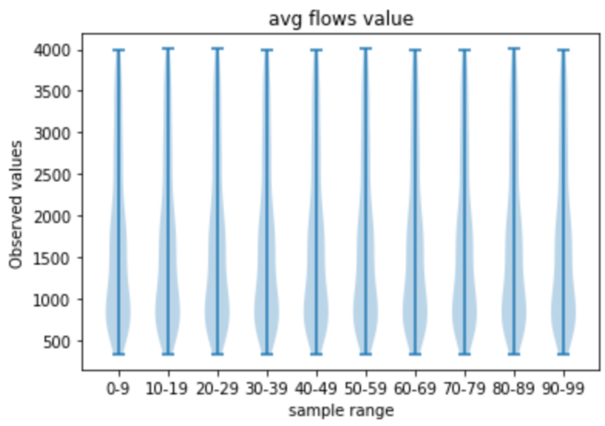
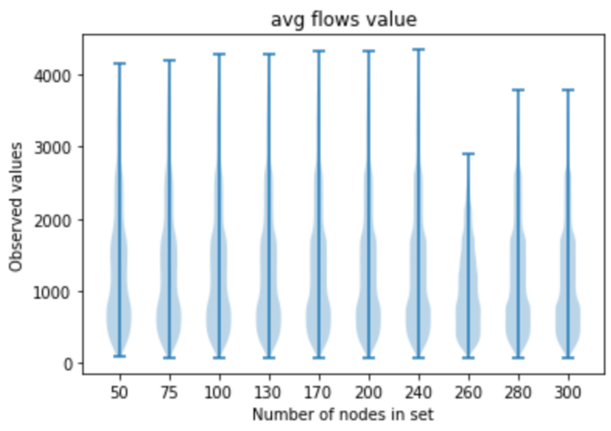


Figure 1: Degree Histogram of our selected network topologies



(a) Traffic flow distribution of the training set



(b) Traffic flow distribution of the validation set

Figure 2: Traffic flows distribution Training vs Validation set

The pseudo codes describing how the network topologies, the traffic matrices and the final dataset have been generated are given in Algorithm 3, Algorithm 4 and Algorithm 5, respectively.

5 Conclusion

This work presents a method to identify and generate the most relevant training samples, in order to reduce the cost of generating datasets in the networking domain. We proposed a three-step approach that results in building a high-quality dataset such that with only 90 samples of 10-node networks, we can train a model that scales effectively to samples of large networks with 50 to 300 nodes.

References

[1] Technical Report: RouteNet-Fermi. `https://bnn.upc.edu/download/technical_report_routenet_fermi`.

[2] Vladimir Batagelj and Ulrik Brandes. "Efficient generation of large random networks". In: *Physical Review E* 71.3 (2005), p. 036113.

Table 3: MAPE (%) obtained on validation sets

Validation set	50 nodes	75 nodes	100 nodes	130 nodes	170 nodes	200 nodes	240 nodes	260 nodes	280 nodes	300 nodes	ALL
MAPE(%)	6.0	6.2	6.8	7.9	8.4	8.4	8.0	3.1	6.1	5.4	6.5

Algorithm 3: generate_topology()

```
Input: num_nodes, prob, policies,
       buffer_sizes,wfq_weights,
       drr_weights
$G = \mathcal{G}_{n,p}$(num_ nodes, prob) [2]
for node in G do
    node_schedulingPolicy = random(policies)
    node_bufferSizes = random(buffer_sizes)
    if node_schedulingPolicy == WFQ then
        wfqWeight = random(wfq_weights)
    if node_schedulingPolicy == DRR then
        drrWeights = random(drr_weights)
```

Algorithm 4: generate_traffic()

```
Input: G, intensity, time_dists,td_weights,
       size_dists, sd_weights, tos_list,
       tos_weights
$traffic$ = matrix of zeros of size G
foreach (src,dst) pair in G do
    $b_{avg}$ = random($[intensity/2, intensity]$)
    $td$ = random(time dists, td weights)
    $sd$ = random(size dists, sd weights)
    $tos$ = random(tos list, tos weights)
    $traffic[src, dst] = b_{avg}$
Return: $traffic$
```

Algorithm 5: generate_dataset()

```
Input: num_nodes, prob, capacity_set,
       policies, buffer_sizes, wfq_weights,
       drr_weights, time_dists, td_weights,
       size_dists, sd_weights, tos_list,
       tos_weights, intensity_set
for i in 1...10 do
    for j in 1...10 do
        G = generate topology()
        $paths$ = shortest paths routing
        $intensity$ = random(intensity_set)
        $traff\_ar$ = generate_traffic()
        set_link_bandwidt()
generate_dataset_simulator() #use omnet++
clean_dataset() # as described in section 4
```