ITU-ML5G-PS-001: Graph Neural Networking Challenge 2021 - Creating a Scalable Network Digital Twin

Miquel Farreras Casamort[1], Paola Soto Arenas[2], Lluís Fàbrega Soler[1], Pere Vilà Talleda[1]

[1]Institute of Informatics and Applications, Universitat de Girona, Girona, Spain
[2]IDLab, University of Antwerp, in collaboration with imec, Antwerpen, Belgium

I. Introduction

The ITU (International Telecommunication Union) proposed a artificial intelligence - machine learning (AI/ML) event, called *ITU AI/ML in 5G Challenge: applying machine learning in communication networks* where different networking problems must be solved by using AI/ML algorithms. The BNN-UPC (Barcelona Neural Networking Center - Universitat Politècnica de Catalunya) proposed the *Graph Neural Networking Challenge 2021. Creating a Scalable Network Digital Twin* challenge. The main goal of this challenge was to improve the scalability of RouteNet, a Graph Neural Network (GNN) model that estimates per-source-destination performance metrics in networks. The baseline RouteNet model, when trained with small networks (25 to 50 nodes) and then used with larger networks (50-300 nodes), resulted in very high errors in the per path average delay (measured with the Mean Average Percentage Error or MAPE). The BNN-UPC provided two initial implementations for solutions, one in Tensorflow, and the other in iGNNition. The GAIN (Girona and Antwerp Intelligence for Networking) decided to use the Tensorflow implementation because of the larger community.

II. Description of the best approach

Our best solution was implemented using the following steps:

1) Apply the queue occupancy approach: As suggested by the challenge GitHub documentation, we started testing the change of the original Routenet Tensorflow implementation, from predicting path delay to predict queue occupancy, and later apply the post-processing to obtain the required path delay. To predict the queue occupancy, we followed these steps:

1) Modify the *read_dataset.py* file and change the prediction to occupancy
2) Change the input of the readout function in the *routenet_model.py* file, as the predicted variable was related to the link and not to the path state.

The final post processing to estimate each path delay from the queue occupancy results was the sum of the queue delays belonging to a path. Each queue delay of this path was computed as $(O_l * Q_s * A_p)/C_o$ where O_l is the occupancy of the queue, Q_s is the queue size in number of packets (*queueSizes* in the dataset), S_p is the average packet size in number of bits (*AvgPktSize* in the dataset) and C_o is the capacity of the outgoing link of the queue (*bandwidth* in the dataset). More details on the algorithm implementation are included in the pseudocode in Algorithm 1.

TABLE I
MAPE original Routenet vs Occupancy implementation

	Full validation	**Setting 1**	**Setting 2**	**Setting 3**
Original model	187,28	79,14	253,07	247,21
Occupancy	28,82	63,58	21,02	21,54
		50 nodes Setting 1	**50 nodes Setting 2**	**50 nodes Setting 3**
Original model		50,15	68,48	44,66
Occupancy		18,86	21,04	17,18
		300 nodes Setting 1	**300 nodes Setting 2**	**300 nodes Setting 3**
Original model		92,97	345,13	368,01
Occupancy		96,41	31,56	24,26

2) Limit occupancy results: In a next step, we detected that some few results predicted a negative or bigger than one occupancy. We corrected this error by setting negative predicted occupancy to zero, and bigger than one occupancy to one.

3) PathLength: We added an extra feature called *pathLength*. As the name suggests, it is the number of nodes the path goes through, including the origin and destination nodes. We noticed in some experiments that our validation dataset results were worse by using subsamples with longer paths. The original results in Table I show that the MAPE with longer paths are the worst case scenario for our model. Full validation is the full validation dataset, Setting 1 refers to longer paths, Setting 2 to larger links and Setting 3 is a mix of both features. So we expected the pathLength attribute to improve the results.

TABLE II
PARAMETERS USED IN THE STRUCTURE OF THE GRAPH NEURAL NETWORK

Link state dimension	16
Path state dimension	32
Message passing iterations	8
Readout units	8
Learning rate	0.001
Epochs	80
Steps per epoch	4000
Validation steps	5

4) Normalize predictor variables: After analysing the train and validation datasets, we observed that the data distributions for the capacity, traffic and *pathLength* were in different value ranges in both datasets. So we applied a min-max normalization as a preprocessing in the *transformation* function (also as suggested for the bandwidth on the challenge *GitHub*).

5) Parameters: No modifications were applied to the loss functions, the hyper-parameters, nor the Routenet architecture. We realized that the number of training epochs could be reduced, as the model was stabilizing the *val_MAPE* earlier than the original 100 epochs. The rest of the parameters were the same as default. Table II shows the used parameters.

An *early stopping* epoch limiter is a technique that stops the training after a number of epochs where the model does not improve its MAPE. The number of epochs monitored to do the early stopping is called *patience*. This approach was tested for other implementations, but not in the best one submitted. The *Weights and Biases* platform was used to easily verify the impact of the different changes in the validation MAPE result.

6) Features: All the features used for paths and links were normalized, as their type was scalar and the value distribution between the training and validation datasets was divergent, as shown in Table III.

TABLE III
FEATURES USED IN THE BEST SOLUTION

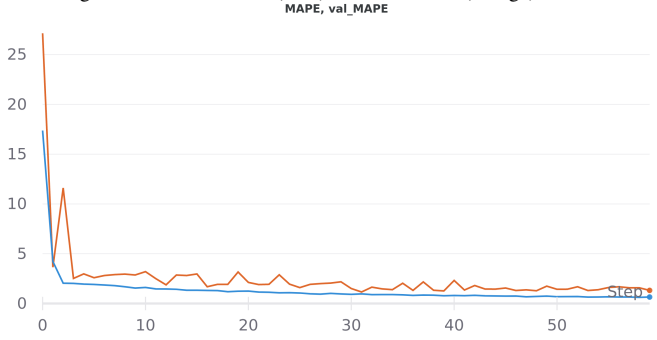
Path features			
Feature	**Definition**	**Min. value (train)**	**Max. value (train)**
Traffic	Average bandwidth through the path	30.787	2048.23
PathLength	Number of nodes in path	2	7
Link features			
Capacity	Link bandwidth (bits/time unit)	10000	100000

III. PROBLEMS AND POSSIBLE IMPROVEMENTS

The solution described in section II was the one that gave the best results in the evaluation platform. Another solution was implemented, including a new feature, called offered traffic to a link. This feature was defined as the sum of the average traffic (feature AvgBw) of all the paths passing through the link divided by the bandwidth (feature capacity) of that link, resulting in a scalar variable assigned to the link state. Moreover in this solution we also included the number of packets generated (pktsGen in the dataset), max-min normalized, as a path feature, as higher correlation to delay was seen in the data analysis. The training achieved a low validation MAPE for the occupancy prediction (1.317 %) on the validation dataset as shown in Figure 1. However, a 47% MAPE for the path delay prediction was obtained on the evaluation platform. It is suspected that there is some problem with the postprocessing of the delay from the occupancy prediction, maybe a bug in the code. Further work will be dedicated to investigate this problem.

Another possible improvement was the calculation of the average packet size for each link, instead of reading the dataset attribute, for higher accuracy (it is supposed to be a calculated, not measured value). This can be obtained from the average bandwidth (AvgBw) in bits per second for each path, divided by the packets generated (PktsGen). Then calculate all the average packet size for each path. And finally, for each link, obtain the average packet size of the paths that go through each link. The packet size used in the occupation to delay postprocessing would be the result calculated in this procedure.

Fig. 1. MAPE on train (blue) and validation (orange) datasets



IV. CONCLUSIONS

As a first experience in GNN for some of the participants, we are satisfied with our result. There were some time and technical limitations during the challenge. We plan to work further on our solution to improve the results once the test dataset with labels is released. We have some ideas related to traffic that we believe will reduce the MAPE. This competition has helped us to gain experience in tuning and developing GNNs, a field that has a steep learning curve. It was a pleasure to contribute to this project and we are looking forward to the next challenge.

ACKNOWLEDGEMENTS

The GAIN team wants to thank the contributions and insights of Dr. Miguel Camelo and the University of Antwerp having given access to us to their GPU platform for machine learning.

Algorithm 1: Delay inference from occupancy

Input: D: list of directories where samples are placed.
O: list of occupancy predictions in order per sample and paths.

Output: Dr: list of delay results for all paths in all samples, sorted by order of samples.

Function calculateDelayFromOccupancy($source$, $destination$):
$\forall flow \in T[source, destination]$['Flows']
if $T[source, destination]$['Flows'][flow]['AvgBw'] != 0 and T[$source$, $destination$]['Flows'][flow]['PktsGen'] != 0
$route$ = $R[source][destination]$
$delayRoute$ = 0
$\forall index, node \in route$
$nextNodeIndex$ = $index$ + 1
if $nextNodeIndex <$ length($route$):
if $S[node][route[nextNodeIndex]] < 0$ then $S[node][route[nextNodeIndex]]$ = 0
else if $S[node][route[nextNodeIndex]] > 1$ then $S[node][route[nextNodeIndex]]$ = 1
$delay$ = $S[node][route[nextNodeIndex]]$*(G.nodes[$node$]['queueSizes']*$T[source$, $destination$]['Flows'][$flow$]['SizeDistParams']['AvgPktSize'])/$G[node][route[nextNodeIndex]]$[0]['bandwidth']
$delayRoute$ = $delayRoute$ + $delay$
return $delayRoute$

Main algorithm:
1) create iterator i for O.
2) $\forall directory \in D$
$\forall sample \in directory$
create empty results R
G = sample topology object
T = traffic matrix object
R = routing matrix object
S = sample occupancy matrix, initialized with zeros and with G.nodes*G.nodes size
$\forall edge \in G$.edges
$S[edge$.source][$edge$.destination] = next i element
$\forall source \in G$.nodes
$\forall destination \in G$.nodes
if $source \neq destination$
d = calculateDelayFromOccupancy($source$, $destination$)
add d to Dr array
3) return Dr, list of delay results for all paths in all samples, sorted by order of incoming samples.