Predicting Queue Network Distributions with Gated Graph Convolutional Neural Networks

Roberto Dailey – Queueing Spring 2022

***Abstract*—Predicting queueing network behavior is a necessary process for designing and operating queueing networks. Recently, Graph Neural Network (GNN) approaches such as RouteNet, have been developed with high success at estimating point estimates of queue performance. However, these models are often limited to point estimates and none so far have provided an estimate of the full state distribution of each queue within the queueing network. In this paper I addresses these limitations by using a Graph Neural Network on generalized queue networks to predict the distribution of the number of jobs for each node in a queueing network.**

All code for this project is available publicly, and can be run in browser with Google Colab, using the following link: https://colab.research.google.com/drive/1GFRM_BggG2EScZVozKPYO2qQ_LfEVm4Z?usp=sharing

***Index Terms*—Queueing Networks, Queue Behavior, Graph Neural Networks, Gated Convolutional Graph Neural Network.**

I. Introduction

Modeling queueing networks is essential for many domains, such as transportation planning, semiconductor manufacturing, and telecommunications. Accurate queueing models allow for optimization and prediction by stakeholders of these networks. Queueing theory offers tractable solutions to a subset of these networks, but when common assumptions such as finite buffers are applied, tractable solutions break down. For these networks, modeling must be done either by simulating the network or using data driven methods/heuristics such as those presented in [1].

Recently multiple studies have taken advantage of advances in Deep Learning on graphs with Graph Neural Networks (GNN's) [2]–[4]. These studies have been successful in predicting queueing network performance after being trained on simulated data. They have even shown evidence of generalizing to topologies and network sizes not seen in the training sets. They are limited however in that they all produce point estimates, not distributions. Their inference is limited to the network parameters they train on, such as delay or average number of customers in each queue.

This paper presents a form of GNN called a Graph Convolutional Neural Network to predict the distribution of the number of jobs/customers in a FIFO single class network queueing system. By modeling the distribution for each node, other time-based parameters can be constructed as needed. These include average queue length, utilization, and percent of time the buffer is full. For networks where arrivals see time averages, job-based metrics such as average delay can be calculated as well. This paper will look at both the accuracy of predicted distribution of each node in a queue network as well as two queue performance measures: portion of time the queue is idle and queue length.

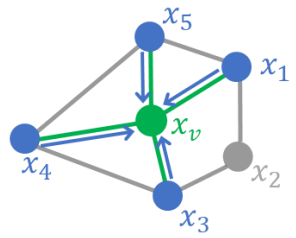
To train these networks, an abstracted queue network simulator was constructed. It is capable of simulating 30,000 network events (services, arrivals) per second. Based off my own search, this is the first publicly available queue network simulator for python that allows for arbitrary topologies and event distributions.

Even at the speed of the queueing simulation, this work was heavily constrained by computing time. Thousands of queueing simulations had to be run to build the training data for the neural networks. Because the data pulled from the simulations was distributional, it converged much slower than standard point estimates, such as the average queue length. Additionally, this analysis was run in Colab to take advantage GPU based neural network training, however this service only maintains an environment 24 hours at a time, so simulations could not be let to run for more than a few hours. Given all these constraints, this project aimed for predicting the queue network distributions at a set time point from the start of the simulation, before the distribution converged to its steady state. This would mean the GNN was not learning steady state distribution, but state distribution at a set time after queues start.

A. Convolutional Graph Neural Networks

Neural Networks are composed of a layer of nodes containing learnable functions. Together these layers of nodes can produce arbitrary multi-dimensional functions when trained on data. These models usually take in a fixed sized input, pass it through a set of hidden layers of learned functions, then produce a fixed sized output. Using a fixed input-output framework would be poor for modeling queueing networks as we wish to model networks with arbitrary numbers of nodes and edges. Modeling each node independently will fail to capture important interactions between queues in the network. Recent advancements such as Graph Convolutional Networks [5] get around these issues by having each layer preform a convolution for each node over the adjacent nodes, with the function

parameters for this convolution being shared by every node in the graph.



$$[1] \quad h_v = f\left(W \cdot \sum_{u \in \mathcal{N}(v)} \frac{x_u}{|\mathcal{N}(v)|} + B \cdot x_v\right)$$

Figure 1/ Equation 1. Description of the graph convolution. h_v is the output of the activation function, f at the current node v. W and B are learnable parameters. $\mathcal{N}(v)$ is the set of adjacent nodes to v. If this is the first layer in the neural network, x_i is a feature from node i, otherwise it is the activation function output from the previous layer.

As shown in Figure 1. The GCN sums features from adjacent nodes with a learned parameter and passes the result to the activation function. This allows information to propagate around the graph within the Graph Convolutional Neural Network. It should be noted that in this formulation, information can only traverse one node per layer, so if two nodes are not adjacent it will take more layers in the GCN to convey information between them. It should also be noted that this process is repeated for each node in the original graph using the same parameters for the convolutions. Using this system, the GCN is flexible to changing number of nodes/edges in a graph. Each node in the queue graph shares the same functions in the GCN while collecting information from adjacent node features/layer activations.

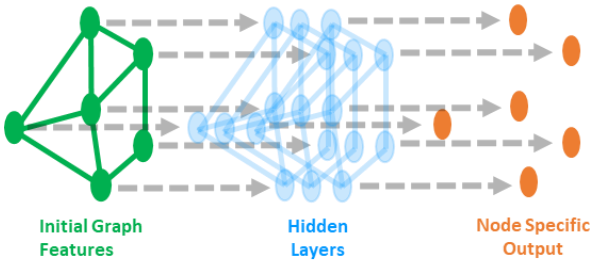


Figure 2. Flow of information through the graph convolutional neural network. Each node uses the same learned parameters to pass information to the next layer. The result is a prediction at each individual node.

II. Methodology

A. *Training Data – Network Queueing Simulation*

To train the model on network parameters, data is needed on queue state distribution over a variety of queue networks. To provide novel data, the queue network data needs to cover some basic objectives:

1. Represent Queues with non-trivial steady state representations (i.e. queues with buffers)
2. Cover a range of network topologies
3. Cover a range of queue parameters (service rate, arrival rate, buffer size, etc.)

Given that a variety of conditions and network structures needed to be tested quickly, a synthetic network queueing simulation was developed so that these parameters could easily be adjusted. For training and testing data, random queueing networks are generated and then simulated to get the state distribution for each queue in each network.

The simulation is event based – a simulation clock advances one event at a time, with events being either services or arrivals.

Buffers are set with a blocking after service (BAS) system: If a queue routes to another queue with a full buffer, it finishes the current service then holds until the full buffer opens a spot. Outside jobs that arrive when a buffer is full will be turned away. If multiple queues route to a full buffer, they fill the buffer in the order which they routed to it. In networks with cycles, this has potential to cause locking, where all queues in the queue network would become stuck. This is a situation we want the GNN to be able to model, so we won't create mechanisms to avoid it.

B. *Queue Simulation Scenarios and Parameter Ranges*

The dataset for training and testing queue parameters needed to cover a comprehensive set of scenarios and topologies to demonstrate the generalizability of the CGN. For each simulated network in the training set, a topology is selected from a set of 4 topologies as described in Figure 4.

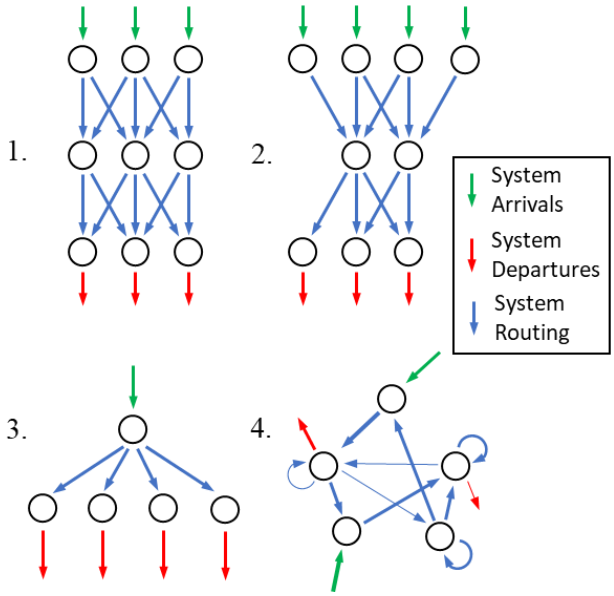


Figure 3. Types of topologies of queue networks generated for the datasets. 1. Sequential, with equal number of queues per row. 2. Sequential with varying number of queues per row. 3. Hub serving a set of queues. 4. Network with random connection weights.

Figure 4. Distribution of truncated average number of jobs over nodes in simulated queueing networks. (Each individual queue node within the queueing networks, not average number of jobs in each whole network.)

Full descriptions of how parameters are selected for each topology are listed in the appendix part A.

In addition to the topology selection, queue networks were given random parameters for buffer size, service rate, and where applicable, arrival rate. The randomization of these parameters was defined to cover a range of behaviors for the queues, with some remaining mostly empty, some behaving at a steady state with modest queue length, some being unstable, and some becoming locked by their buffers.

To ensure these goals were matched, the distribution of average jobs in each queue system was averaged over 200 generated networks for each topology, with parameters matching the training dataset. These distributional results are shown above in figure 4. One item to note is for this analysis is the distribution for the number of jobs is only calculated up to Q = 10 jobs. For this analysis all work was done with truncated distributions, any time that simulated queues spent with Q>=10 was added to the bin for Q = 10. I will expand on this decision in later sections.

C. *Training Data -16,000 network simulations*

Training data was built by running 16,000 queue network simulations, 4,000 simulations per topology type. Networks were generated with a random number of nodes ranging from 3 to 20 nodes. From my own testing, the GNN would have trouble improving testing accuracy with smaller training sets. I believe even larger training sets would still provide great room for improvement of the model. As mentioned in the introduction, these networks were limited by training time. Each network was stopped after t = 800 seconds, at which point its distribution of system jobs was measured as:

$$[2] \qquad p(Q = x_i) = \frac{1}{t} \sum_{e_k \; s.t. \; Q(e_k)=x_i} e_k$$

Where e_k are the queue network simulation inter-event times, Q is the number of customers in the individual queue node (not the whole network), and x_i is the bin of the distribution being estimated from Q = 0 to Q>=10.

D. *Queue Simulation Verification with Open Jackson Network*

To test that the queue simulation was working correctly, a quick test was done to compare the simulation to analytically tractable settings. Initial queue network parameters were set up to mimic an Open Jackson Network: service and arrival distributions were set to be exponential, while buffers for each queue were set to infinite length. The simulation was let to run until its queue distributions converged, then its individual node distributions were compared with the expected distribution for each node of an equivalent Jackson network:

$$[2] \qquad p(x_i) = (1 - \rho_i)\rho_i$$

Here ρ_i is the utilization, (λ_i / μ_i) for node I, with λ_i being the effective arrival rate for node i and μ_i being the service rate. The vector of λ_i's is calculated as:

$$[3] \qquad \lambda = (I - P^T)^{-1}\alpha$$

Where P is the routing matrix, and alpha is the vector of exterior arrival rates.

Figure 4 shows one of the nodes from the tested queue network, with the simulation queue distributions matching the expected Jackson network distribution almost exactly. Additionally, the same network with a buffer size of four on this node is shown, and its effect on the state distribution can clearly be seen. One item to note on figure 4 is the increase in probability of the queue having 10 jobs. This is the tail end mass of the distribution being added up and placed in bin 10.

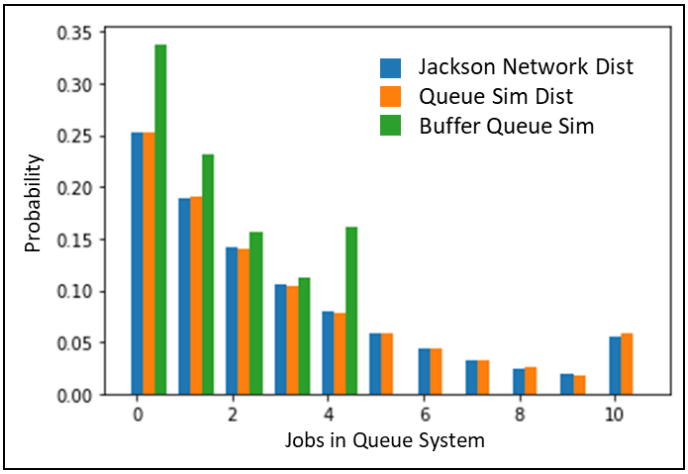


Figure 6. Comparison of analytical solution to Open Jackson Network with queuing simulation results. This is also compared with a blocking after service queueing system with a buffer of length 4 on this queue. Additional node distributions and network parameters are listed in the appendix part B.

E. Neural Network Model - Residual Gated Graph Convolutional Neural Network

For a Graph Neural Network to learn queueing behavior, it needs to pull information from both adjacent node features as well as routing between nodes (i.e. edge feature information). For this to work, a modification on Graph Convolutional Neural Networks is needed to learn information from edge data. For this a Residual Gated Graph Convolutional Neural Network was used, based off the work of [6]. The modifications from a standard Graph Convolutional Neural Network are shown in the three equations below:

$$[4] \qquad h = x + \left(Ax + \sum_{v_j \to v} \eta(e_j) \odot Bx_j \right)^{+}$$

$$[5] \qquad \eta(e_j) = \sigma(e_j) \left(\sum_{v_k \to v} \sigma(e_k) \right)^{-1}$$

$$[6] \qquad e_j = Ce_j^x + Dx_j + Ex, \quad e_j^h = e_j^x + (e_j)^{+}$$

Equations 5-6. Description of the Residual Gated GCN activation. Equation 2 here defines the activation of GCN node h based on x (the feature at current node v) as well as a multiplication of incoming edge activations, $\eta(e_j)$, with adjacent node features x_j, multiplied by a set of learnable parameters B. *All Matrices highlighted red* are learnable parameters. Equation [3] defines the normalization for edge activations, with sigma being a logit function. Equation [4] defines the edge activation as well as hidden edge representation. For hidden layers, the features e_j^x and x_j are replaced by activations from previous layer, e_j^h and h_j, respectively.

The inputs to the GNN used for this project are service rate, arrival rate, and buffer size as shown in figure 7 below. For this proof of concept, the network predicted 11 outputs, from 0 to 10 jobs in an individual node. When distributions were calculated, any probability mass past 10 was summed and added to the bin for 10. While this greatly simplifies analysis and network construction, this also limits model prediction to queue behavior within 10 jobs in the system. Future work could look at addressing this through other distribution estimation methods, such as estimating distribution percentiles.

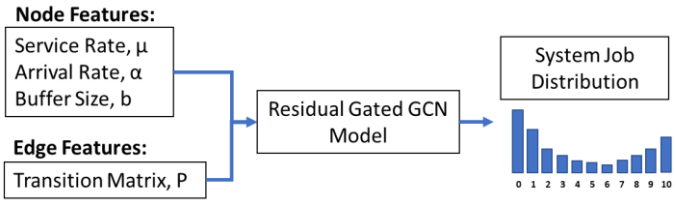


Figure 7. Inputs and outputs to Graph Neural Network trained on simulated queue network data.

The GNN prediction model was built with 10 of the Residual Gated Graph Convolution Layers, with 60 nodes per layer. A final fully connected relu layer translated to the 11-dimensional output per node – the predicted state distribution. Edge connections existed for all adjacent nodes with routing probability greater than 0. The GNN trained for 2500 epochs, with loss being shown in figure 5.

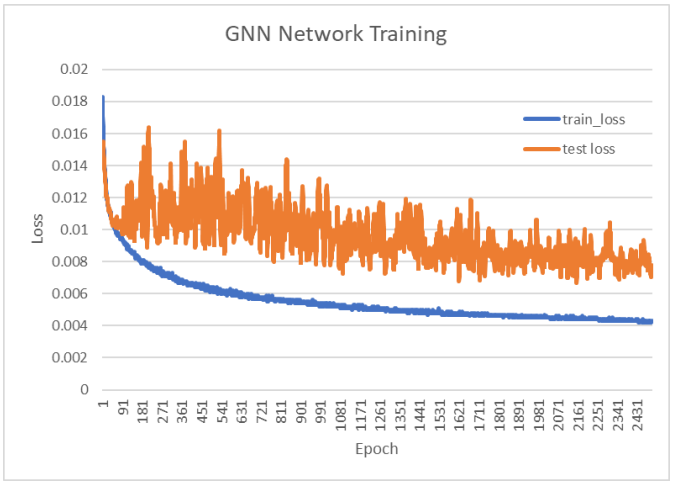


Figure 5. *Test/Train loss as the GNN network trained.*

III. Results

A. Predicting Distribution of Number of Jobs

The GNN prediction model was tested on networks with sizes ranging from 3 to 40 nodes for each topology. Each size/topology combination had 120 random networks not previously seen by the GNN during training. The GNN had seen networks between sizes of 3 and 20 in the training set, while larger networks of size 30, 40 were meant to test the GNN's ability to abstract to larger networks. GNN probability estimates for each node were compared with simulation results, and Root Mean Squared Error (RMSE) was taken, shown below in table 1. Additionally, the coefficient of determination (1-RSS/TSS) was determined for each test. TSS was calculated separately for each pair of network size/ topology.

RMSE of p(Q = x)

# nodes	top 1	top 2	top 3	top 4
3	0.080	0.068	0.062	0.069
7	0.074	0.074	0.044	0.094
10	0.067	0.073	0.049	0.095
20	0.062	0.075	0.043	0.129
30	0.069	0.084	0.096	0.146
40	0.066	0.082	0.138	0.183

Table 1. RMSE of the GNN predictions of truncated distributions of the number of customers in each queue system up to time t = 800s.

R^2 of p(Q = x)

# nodes	top 1	top 2	top 3	top 4
3	0.81	0.88	0.91	0.91
7	0.85	0.87	0.95	0.82
10	0.87	0.87	0.94	0.82
20	0.88	0.87	0.95	0.65
30	0.85	0.85	0.77	0.56
40	0.87	0.86	0.51	0.27

Table 2. Coefficient of determination of the GNN predictions for the truncated distributions of the number of customers in each queue system up to time t = 800s.

As can be seen from tables 1 and 2 above, the GNN preformed best on the first 2 topologies, then struggled with topology 4, the random topology. This is the opposite of what I expected. Given that in topologies 1 and 2, information in the GNN can only move one "row" at a time per layer, I expected the model to struggle predicting behavior when arrival rates at the initial row could have a large impact many rows down the queue network. Instead, the GNN preformed well on these networks, and was even able to generalize well to network sizes not seen in training. My best guess for what is happening here is just a general difficulty in predicting topology 4 networks. Because each node is technically connected to every other node, the long run steady state doesn't exist for this topology. As soon as there is 1 finite buffer in the queueing network, the long run probability of the network locking is 1. Given the potential absorbing states, it may be difficult to anticipate how much time until these networks will lock and thus difficult to predict their state distribution at the time t = 800 seconds.

Examples of GNN distribution predictions are shown in figures 8-10.

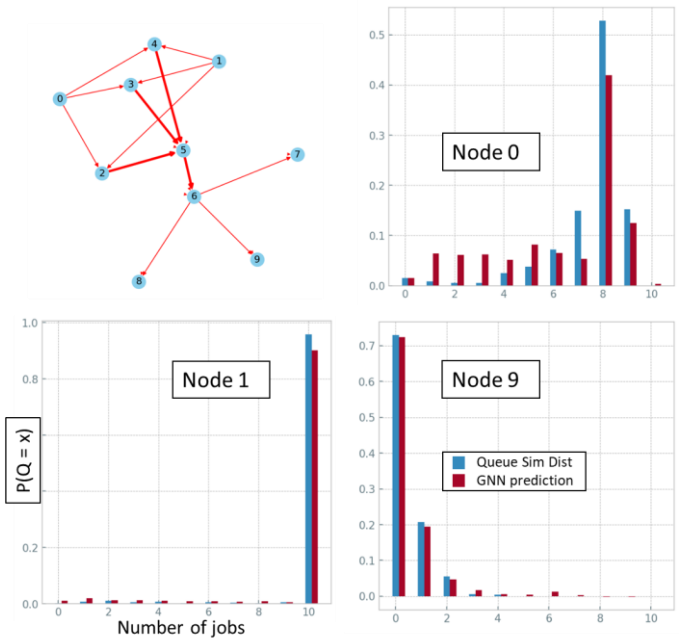


Figure 8. Network of topology type 2, along with distributional predictions for nodes 0,1, and 9. Notice node 1 has all weight on Q>=10, implying this node is either unstable or has a very high average queue length. Given the behavior on nodes 0 and 1, we can see the GNN is able to predict non-Jacksonian system distributions with reasonable accuracy here.

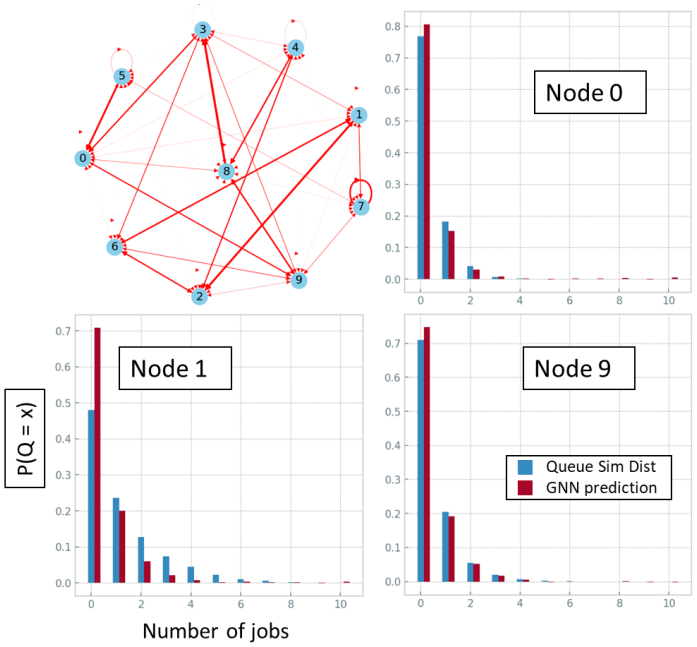


Figure 9. Network of topology type 4 with distributional predictions, with visual edge weights corresponding to routing probability. We can see the GNN underestimating queue lengths on node 1.

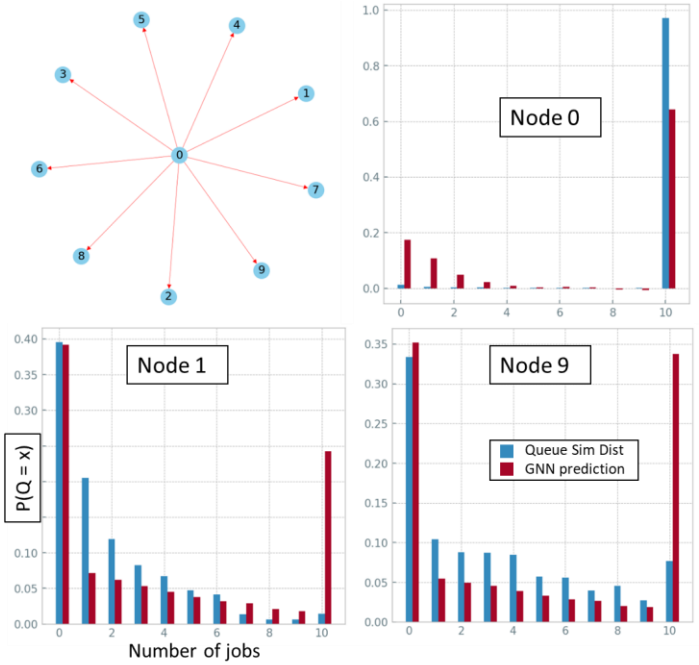


Figure 10. Network of topology type 3 with distributional predictions. Here we see an example of poor prediction quality, as the GNN strongly over-predicts the queue lengths of nodes 1 and 9.

B. Point estimates for portion of time queue is idle and average number of jobs at a node.

RMSE of P(Q = 0)

# nodes	top 1	top 2	top 3	top 4
3	0.078	0.069	0.066	0.068
7	0.063	0.064	0.050	0.072
10	0.057	0.066	0.057	0.072
20	0.050	0.076	0.049	0.117
30	0.051	0.086	0.089	0.129
40	0.050	0.074	0.232	0.171

Table 3. RMSE of GNN prediction of probability each queue system is empty.

R^2 of P(Q = 0)

# nodes	top 1	top 2	top 3	top 4
3	0.84	0.87	0.91	0.91
7	0.90	0.91	0.95	0.90
10	0.93	0.91	0.94	0.90
20	0.94	0.87	0.96	0.72
30	0.94	0.85	0.87	0.67
40	0.94	0.89	0.12	0.39

Table 4. Coefficient of determination of GNN prediction of probability each queue system is empty

Tables 3 and 4. show predictions of the portion of time nodes are idle. This is heavily connected to utilization, with the caveat that in cases where queues are locked due to routing to a queue with a full buffer, the system is not empty, but the server is not technically busy. Patterns in these results match that of the initial distributional results, although GNN was a bit more accurate in predicting this probability bin across different network sizes/types.

RMSE of L

# nodes	top 1	top 2	top 3	top 4
3	1.45	1.06	1.24	1.44
7	1.38	1.23	0.81	2.08
10	1.25	1.18	0.91	2.00
20	1.17	1.14	0.80	2.57
30	1.28	1.31	1.37	2.37
40	1.18	1.26	2.22	2.64

Table 5. RMSE of GNN prediction of average number of jobs in each node.

R^2 of L

# nodes	top 1	top 2	top 3	top 4
3	0.80	0.90	0.85	0.85
7	0.80	0.86	0.91	0.59
10	0.81	0.84	0.87	0.58
20	0.78	0.81	0.88	0.30
30	0.70	0.72	0.52	0.36
40	0.70	0.70	-0.24	0.26

Table 6. Coefficient of determination of GNN prediction of average number of jobs in each node.

Tables 5-6. shows prediction of average number of jobs at each node(L). For Topologies 1-3, the network can almost predict averages within 1 job, however, we see the same pattern as with the probability distributions and queue idle portions where the GNN preforms poorly on topology type 4.

IV. Conclusions/Future Work

This paper was able to demonstrate that Graph Neural Networks can make distributional predictions of queue networks. In sequential networks that mimic assembly lines, GNN's could explain >80% of the variance in probability even for networks twice as large as seen in training. For other network types, GNN's still generally predicted >80% of the variance on test networks of similar size to training. This opens a range of applications where predicting behavior of multiple queue networks is important, but time constraints don't allow for simulation (such as optimization problems).

While this paper was able to produce successful model predictions for certain network queues, it was still inherently limited by its methodology. By using simulation data as training data, it is inherently difficult to test robustness and unlikely events as the simulations would be run for limited amounts of time. This could possibly be addressed with importance sampling as mentioned in [7]. Additionally, this work was limited to predicting job distributions within the 0-10 range. Additional modifications to the GNN would need to be done to predict queues with larger numbers of jobs.

Further work could be done to address the limitations in predicting topology 4, the randomly routed networks. It is possible with a larger set of training data of that type of network, improvement could be seen. It is also possible that other GNN frameworks would need to be adopted, such as attention based GNN's.

V. References

[1] M. Manitz, "Queueing-model based analysis of assembly lines with finite buffers and general service times," *Comput Oper Res*, vol. 35, no. 8, pp. 2520–2536, Aug. 2008, doi: 10.1016/j.cor.2006.12.016.

[2] K. Rusek and M. Drton, "Fine-grained network traffic prediction from coarse data," Jan. 2022, [Online]. Available: http://arxiv.org/abs/2201.07179

[3] M. Ferriol-Galmés *et al.*, "RouteNet-Erlang: A Graph Neural Network for Network Performance Evaluation," Feb. 2022, [Online]. Available: http://arxiv.org/abs/2202.13956

[4] A. Chowdhury, G. Verma, C. Rao, A. Swami, and S. Segarra, "Unfolding WMMSE Using Graph Neural Networks for Efficient Power Allocation," *IEEE Trans Wirel Commun*, vol. 20, no. 9, pp. 6004–6017, Sep. 2021, doi: 10.1109/TWC.2021.3071480.

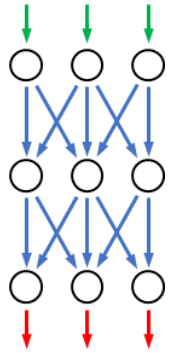
[5] T. N. Kipf and M. Welling, “SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS.”
[6] X. Bresson and T. Laurent, “Residual Gated Graph ConvNets,” Nov. 2017, [Online]. Available: http://arxiv.org/abs/1711.07553
[7] P. Heidelberger, “Fast Simulation of Rare Events in Queueing and Reliability Models,” New York, Jan. 1995. doi: 10.1145/203091.203094.

VI. Appendix

A. List of Network Topology Parameter Generation Settings

List of Network Topology Parameter Generation settings.

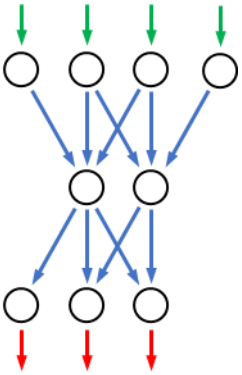
Topology 1: Sequential Stages:

1. 

This first topology was constructed to represent assembly lines with similar capacity at each state of the line.

The width of all rows is set to a random number between 2 and 5. The first row contains arrivals. Arrival rates are set to uniformly distributed between 0 and 1.6. Service rates are uniformly distributed between .4 and 2. Finite buffers are assigned to each node with probability 0.6. If a node is assigned a finite buffer, its buffer size is randomly sampled between 1 and 10.

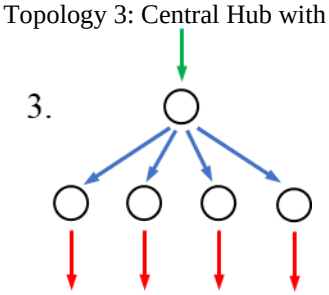
Topology 2: Sequential Stages, variable stage size

2. 

This second topology was constructed to represent assembly lines with bottlenecks occurring at some steps in the process.

The number of queues in each row is a random number between 1 and 5. The first row contains arrivals. Arrival rates are set to uniformly distributed between 0 and 1.6. Service rates are uniformly distributed between .5 and 1.5. Finite buffers are assigned to each node with probability 0.6. If a node is assigned a finite buffer, its buffer size is randomly sampled between 1 and 10.

Topology 3: Central Hub with n users

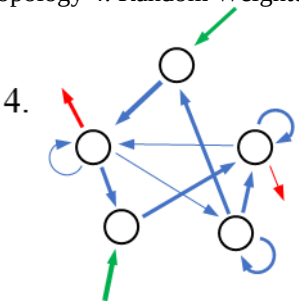
3. 

This second topology was constructed to represent jobs that come through a central server, and then assigned based on their type. (The network only has one job type, but by enforcing assignment to a specific node from the central server, it functions similarly)

The arrival rate for the central node is 1*the number of nodes in the network. (In order to scale so edges of networks still see meaningful jobs at larger network sizes)

. Service rates are uniformly distributed between .8 and 1.2. Finite buffers are assigned to each node with probability 0.2. If a node is assigned a finite buffer, its buffer size is randomly sampled between 1 and 10.

Topology 4: Random Weighted Network.

4. 

Topology 4 was created to represent more random networks that are highly connected. For each node, routing to every other node is selected from a Dirichlet distribution with a uniform alpha of .1. Because of the high amount of connection and possible recirculation, service rate scales linearly with the number of nodes. Service rates are chosen from random uniform distribution ranging from 0.3*number of nodes to 0.5*number of nodes. Arrival rates are sampled from a random uniform distribution from 0 to 0.2. Therefore, jobs can arrive at the system from any node, and depart from any node. Finite buffers are applied to each node with probability 0.4. If a node is assigned a buffer, its buffer size is randomly sampled between 1 and 10.

B. Jackson Network Test Results

Columns correspond to nodes 1,2,3,4 respectively.

The Routing Matrix is listed as below:

0.18	0.30	0.19	0.33
0.12	0.07	0.33	0.43
0.02	0.41	0.26	0.13
0.29	0.23	0.03	0.26

Arrival rates to this matrix are:

0.16	0.11	0.10	0.17

Service rates are:

1.72	1.63	1.66	1.65

Buffer sizes are:

inf	4	inf	inf

Distributions are as follows: