ITU AI/ML IN 5G CHALLENGE 2022

ML5G_PS_002 Graph Neural Networking Challenge 2022

Improving Network Digital Twins through Data-centric AI

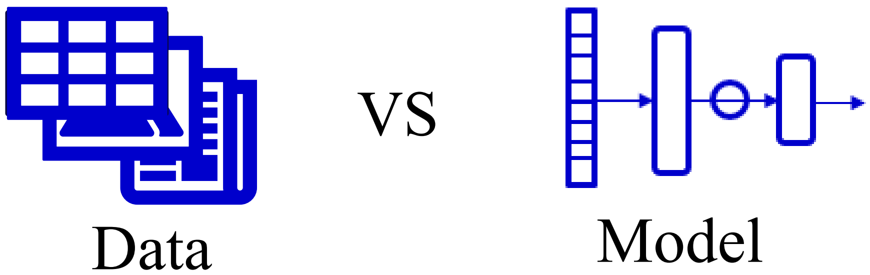
Snowyowl Team

B. Jaumard, J.M. Ziazet, C. Boudreau, O. Delgado

Concordia University, Montreal, Canada

Overview

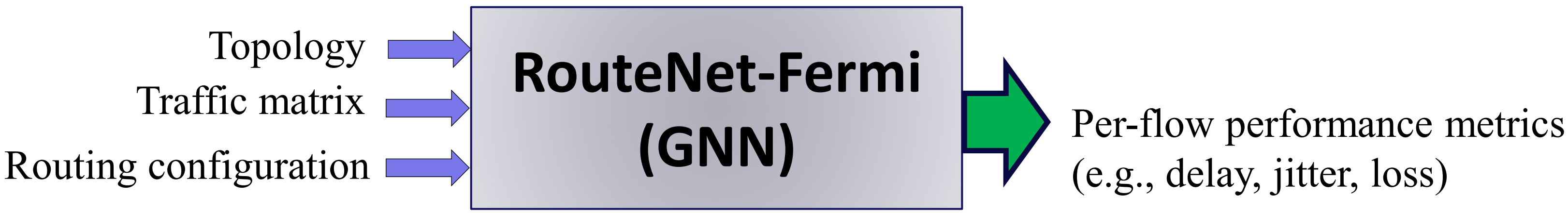
- Deep learning models are traditionally designed along a Model-Centric Approach
  - Focus is on the design of the learning model
  - Acquire as much data as possible, more is better
- Recently, more attention has been directed to the data itself
  - The data used to train the model can greatly influence model performance
    - Quality is important, more is not necessarily better
- Creating appropriate data is often the biggest challenge for developing and deploying AI



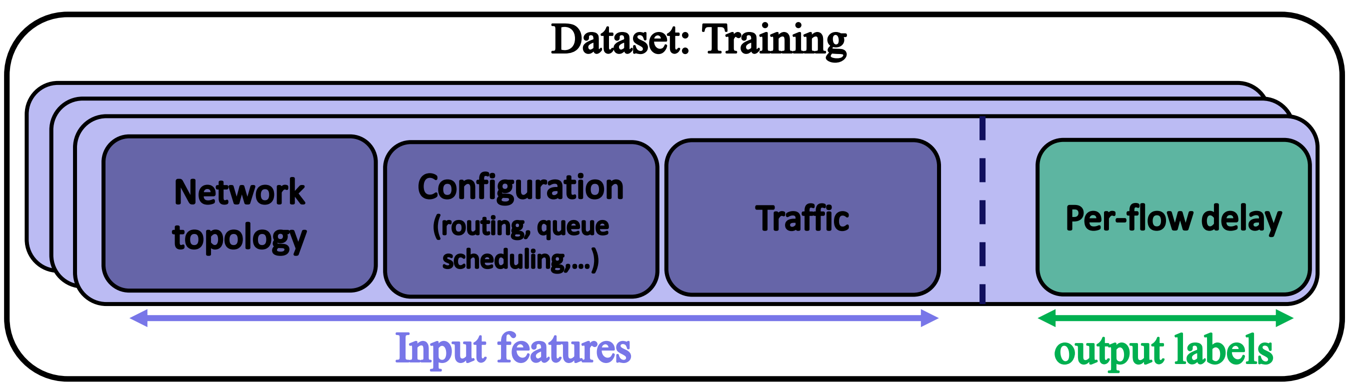
UNIVERSITÉ Concordia UNIVERSITY

Problem Statement

- **Given:**
  - A state-of-the-art GNN model for performance evaluation

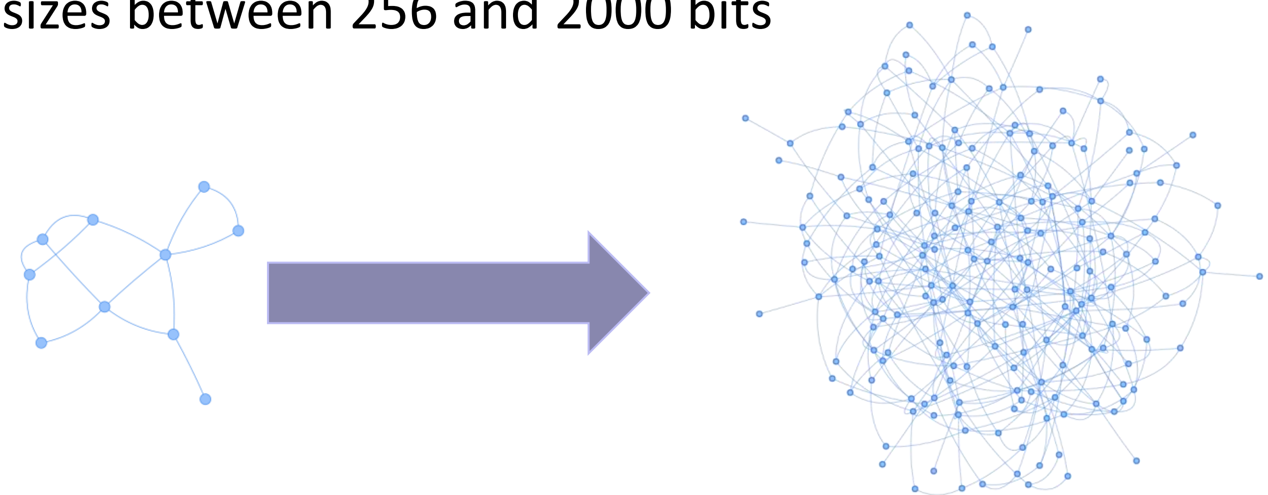


- **Task:**
  - Generate the best dataset to train the GNN model



Constraints

- Maximum of 100 samples (very limited dataset)
- Samples from networks up to 10 nodes (small)
- Bidirectional Links
- Buffer size between 8000 and 64000 bits
- Link bandwidth between 10000 and 400000 and in multiples of 1000
- Average bandwidth between 10 and 1000
- Maximum one path for each source destination pair
- Packet sizes between 256 and 2000 bits
- …



Train
(small networks, up to 10 nodes)

Validation
(large networks, up to 300 nodes)

Solution Overview

A Three-step process

Initial Dataset Generation :
To match validation dataset

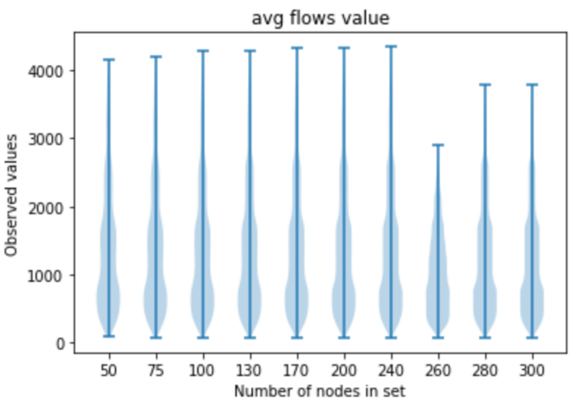
→

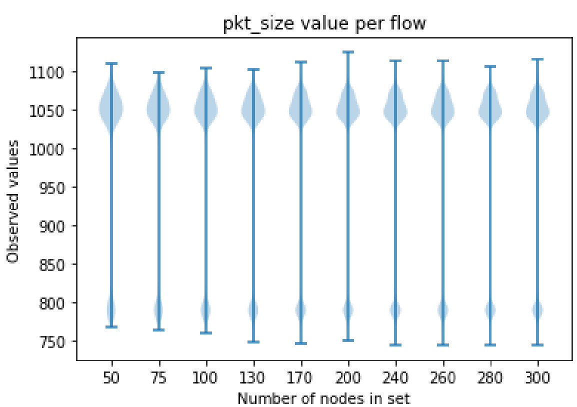
Dataset Refactoring:
To cover use cases of validation dataset

→

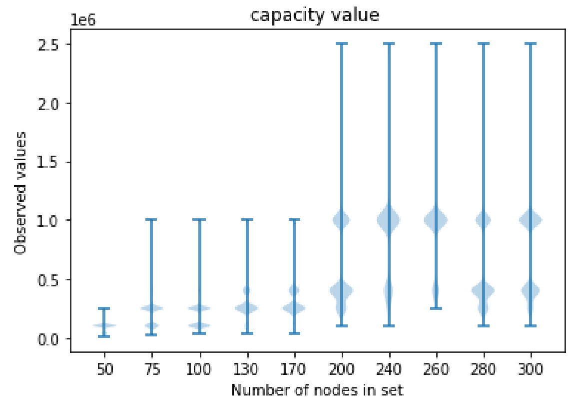
Dataset Cleaning:
To keep only high quality data

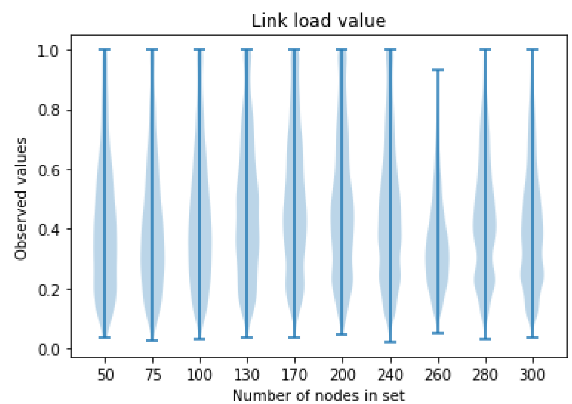
Initial Dataset Generation



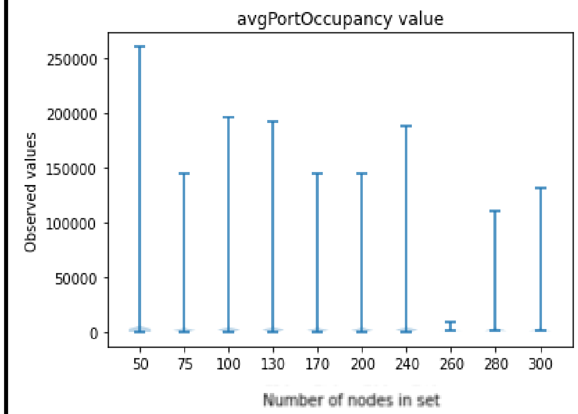


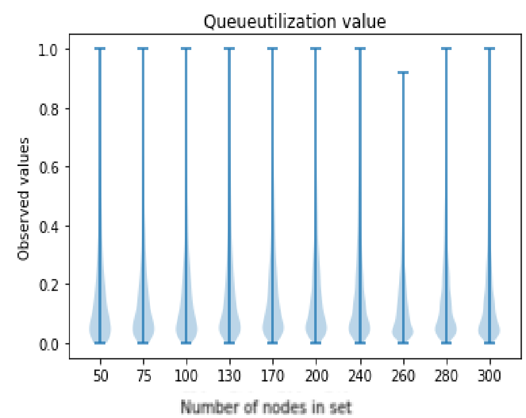
Flow level statistics





Link level statistics





Node level statistics

UNIVERSITÉ Concordia UNIVERSITY

Initial Dataset Generation

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

Set generation parameters that are invariable to network size

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]

Initial Dataset Generation

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

Set generation parameters that are invariable to network size

Need more attention!!!

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]

Initial Dataset Generation :

To match validation dataset

→

Dataset Refactoring:

To cover use cases of validation dataset

→

Dataset Cleaning:

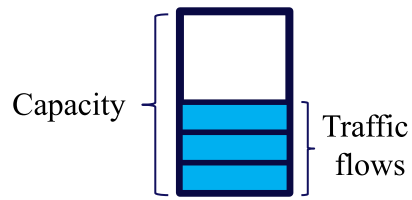
To keep only high quality data

	Updates	Dataset 1	Dataset 2	Dataset 3
1	Initial dataset generation	✓		
2	Dataset refactoring			
3	Dataset cleaning			
	VAL MAPE	~ 27%		

Dataset Refactoring: Link Capacity

1. Set link capacity such that link utilization is variable enough to cover different use cases (different link utilization level)

$$\text{Capacity}_l = \frac{\sum_{f \in N_\ell} t_f}{\text{link\_utilization}_l},$$



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**
  μ = random(mean, weights)
  $\sigma = \mu/2$
  link_load = $\mathcal{N}(\mu, \sigma)$
  if 0 <= *link_load* <= 1 **then**
    break
Return: link_load

2. Set link capacity to be the nearest capacity from a subset of the validation set

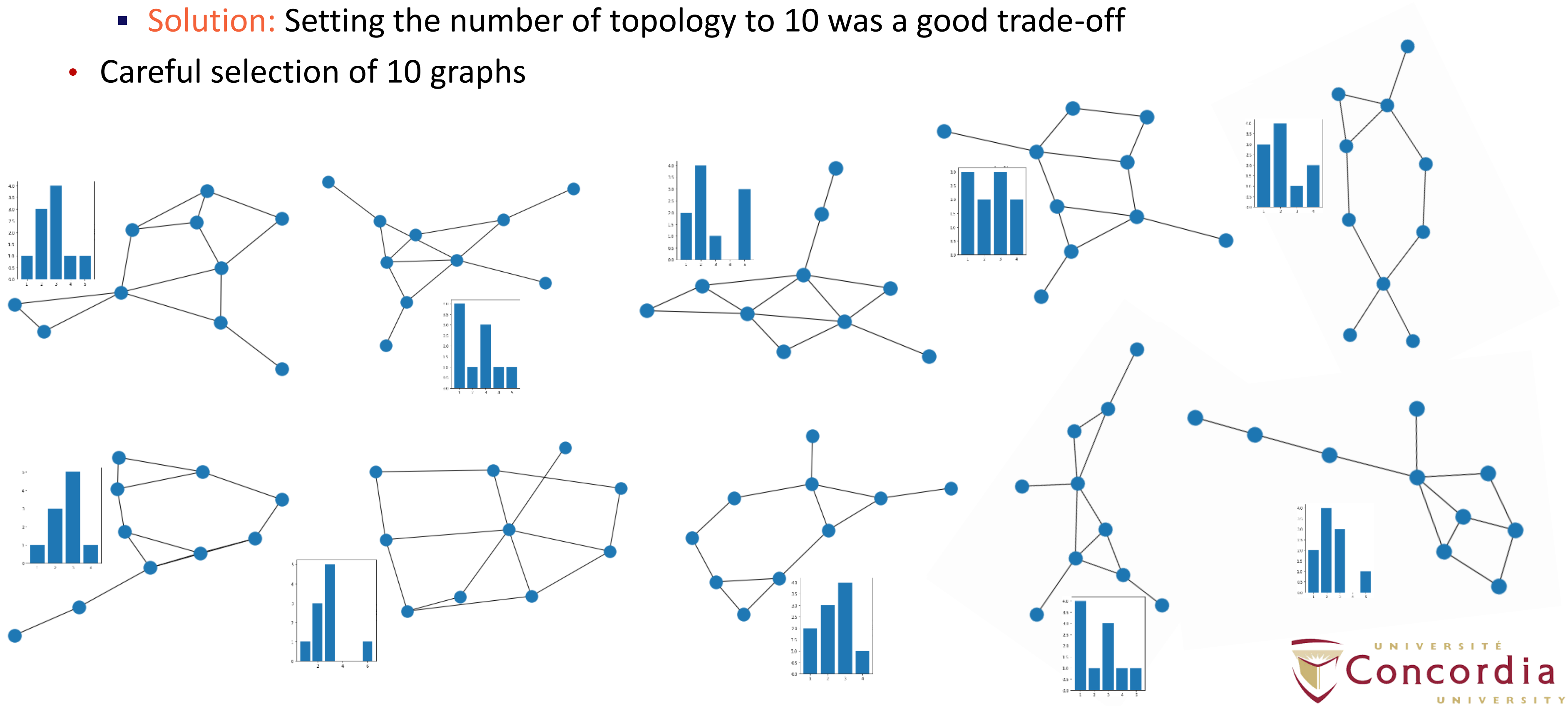
for each $f \in F$ **do** ▷ Flow: Readout
  $\hat{y}_{f_d} = 0$ ▷ Initializing the flow delay
  for each $(q, l) \in f$ **do**
    $\hat{d}_q = R_{f_d}(\boldsymbol{h}^T_{f,l})/\boldsymbol{x}_{l_c}$ ▷ Queueing delay
    $\hat{d}_t = \boldsymbol{x}_{f_{ps}}/\boldsymbol{x}_{l_c}$ ▷ Transmission delay
    $\hat{d}_{link} = \hat{d}_q + \hat{d}_t$
    $\hat{y}_{f_d} = \hat{y}_{f_d} + \hat{d}_{link}$ ▷ Sum of link delays along the flow

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

Dataset Refactoring: Network topology

- Use only 10-node graphs
- 10 graphs vs 100 graphs
  - Observation: Increase the number of network topologies, results in an increase of the variability or noise which negatively affect the results (given the fixed number of epochs).
  - Solution: Setting the number of topology to 10 was a good trade-off
- Careful selection of 10 graphs



Dataset Refactoring: Network topology

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**

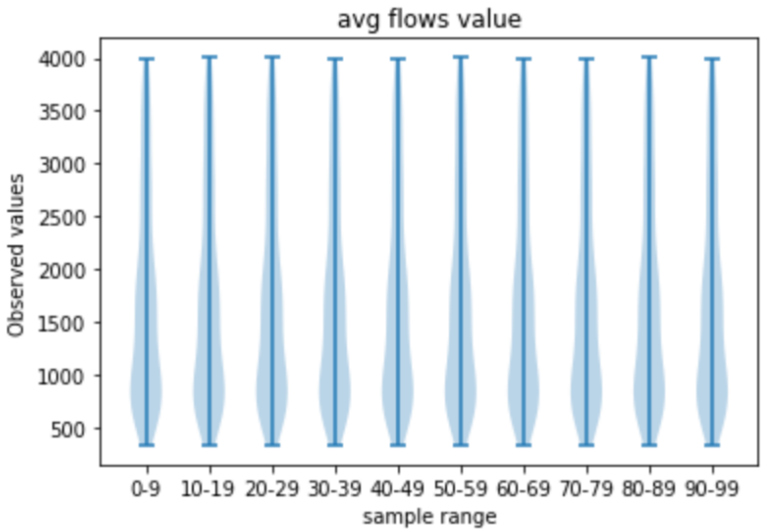
    wfqWeight = random(wfq_weights)

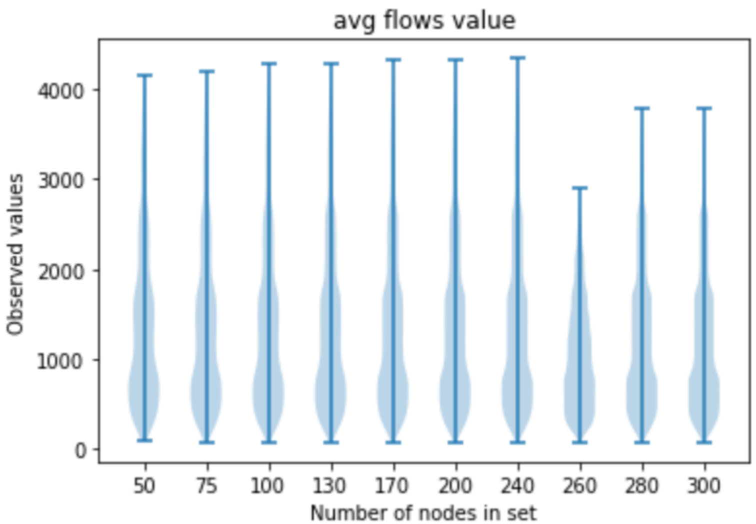
  if *node_schedulingPolicy == DRR* **then**

    drrWeights = random(drr_weights)

Dataset Refactoring: Flow Generation

- 10 traffic matrices vs 100 traffic matrices
  - Observation: Increase the number of traffic matrices, results in an increase of the variability or noise which negatively affect the results
  - Solution: Setting the number of traffic matrix to 10 was a good trade-off
    - 10 traffic flow matrices per network graph and the same traffic flows are successively assigned to flows of other graphs
- Update probability distribution of packet size and time to better match the validation set





Dataset Refactoring: Flow Generation

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**

- $b_{avg} = \text{random}([intensity/2, intensity])$
- $td = \text{random}(\text{time dists, td weights})$
- $sd = \text{random}(\text{size dists, sd weights})$
- $tos = \text{random}(\text{tos list, tos weights})$
- $traffic[src, dst] = b_{avg}$

Return: $traffic$

Initial Dataset Generation :
To match validation dataset

→

Dataset Refactoring:
To cover use cases of validation dataset

→

Dataset Cleaning:
To keep only high quality data

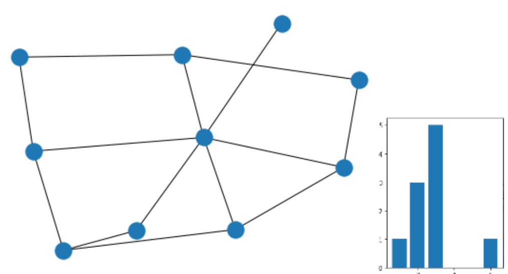
	Updates	Dataset 1	Dataset 2	Dataset 3
1	Initial dataset generation	✓	✓	
2	Dataset refactoring		✓	
3	Dataset cleaning			
	VAL MAPE	~ 27%	~ 8%	

Dataset Cleaning

- Some generated samples may negatively affect performance
- Noise Hypothesis:

Hypothesis	Definition	Why?	Results	Comments
H1	A sample is noisy if it contains averagePortOccupancy that are out of distribution of that of the validation set	The model outputs the averagePortOccupancy our sample distribution should match that of validation set	Not conclusive	
H2	A sample is noisy if it does not contain at least one path of length 4 which represents approximately the average path length in the validation set.	With 8 iterations in the message passing scheme, samples with shorter paths might be more susceptible to over-smoothing	Conclusive	10 samples removed

Removed graph:
Maximum Shortest Path Length: 3
Max Degree: 6



Summary of Results

	Updates	Dataset 1	Dataset 2	Dataset 3
1	Initial dataset generation	✓	✓	✓
2	Dataset refactoring		✓	✓
3	Dataset cleaning			✓
	VAL MAPE	~ 27%	~ 8%	~ 6%

- **Baseline:** Thousands of samples of networks up to 10 nodes, MAPE < 5%
- Our Solution: 90 samples of networks up to 10 nodes, MAPE ~ 6%

Conclusions

- Our findings
  1. Defining the desired level of congestion of the links and deriving capacity from it can be beneficial.
  2. Having too many network topologies is not necessarily beneficial, it is important to find a trade-off and choose topologies that are diverse enough.
  3. Having different traffic patterns are not necessarily beneficial, it is important to find a trade-off and repeating some traffic patterns can be beneficial
  4. Understanding how the GNN model works can help eliminate samples that negatively affect the performance.

- Conclusion:
  - We were able to build a high-quality dataset such that with only 90 samples of 10-node networks, we can scale effectively to samples of larger networks

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.

Thank You!

ANY QUESTIONS?