# GENERALIZABLE POLICY LEARNING USING GRAPH NEURAL NETWORKS: SUPPLEMENTARY MATERIALS

## A Algorithm

Algorithm 1 summarizes our complete training procedure, incorporating the variance reduction and replay buffer techniques discussed in the main paper.

**Algorithm 1** Network Performance Gradient Algorithm

**Require:** Network instances $\mathbf{I}_{\text{train}} = \{I^{(1)}, \ldots, I^{(\text{Num\_Envs})}\}$, learning rate $\alpha$, samples per instance $S$, replay buffer capacity MAX_BUFFER, number of episodes NUM_EPISODES
**Ensure:** Trained model parameters $\mathbf{\Theta}$

Initialize model parameters $\mathbf{\Theta}$, $\sigma$
Initialize replay buffer $\mathcal{D} \leftarrow \emptyset$
Initialize baselines $\eta_\pi(I^{(m)}) \leftarrow 0$ for all $m = 1$ to Num_Envs
**for** episode $= 1$ to NUM_EPISODES **do**
  {Collect experiences through simulation}
  **for** $m = 1$ to NUM_ENVS **do**
    Sample $\{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(S)}\} \sim p^{\pi}_{\theta,\sigma}(\cdot|I^{(m)})$
    **for** $s = 1$ to $S$ **do**
      Evaluate utility $\hat{U}(I^{(m)}, \pi, \mathbf{W}^{(s)})$
    **end for**
    Update baseline: $\eta_\pi(I^{(m)}) \leftarrow (1-\gamma)\eta_\pi(I^{(m)}) + \gamma \cdot \frac{1}{S}\sum_{s=1}^{S} \hat{U}(I^{(m)}, \pi, \mathbf{W}^{(s)})$
    Store $\{(\mathbf{W}^{(s)}, \hat{U}(I^{(m)}, \pi, \mathbf{W}^{(s)}), I^{(m)})\}_{s=1}^{S}$ in $\mathcal{D}$
    **if** $|\mathcal{D}| >$ MAX_BUFFER **then**
      Remove oldest experiences from $\mathcal{D}$
    **end if**
  **end for**
  % Update model using collected experiences
  **for** epoch $= 1$ to NUM_EPOCHS **do**
    **for** $m = 1$ to NUM_ENVS **do**
      Retrieve buffer $\mathcal{D}^{(m)} \subseteq \mathcal{D}$ for $I^{(m)}$
      Compute gradient estimate $\hat{\nabla}_\theta J^\beta$ using replay buffer and baseline
      Update parameters: $\theta \leftarrow \theta + \alpha\hat{\nabla}_\theta J^\beta$
    **end for**
  **end for**
**end for**
**return** $\theta$

### A.1 Hyperparameters

## B Queueing Network Simulation Details

Here we provide details on the queueing network simulation used to evaluate the methodologies proposed in the main paper. The accompanying code can also be found in the codebase.

Table 1: Hyperparameters for Utility Gradient Training Algorithm

| Parameter | Value |
|---|---|
| $\alpha$ | 0.001 |
| $S$ | 5 |
| NUM_ENVS | 10 |
| MAX_BUFFER | 150 |
| NUM_EPISODES | 300 |
| PPO clipping parameter $\epsilon$ | 0.2 |
| $\gamma$ | 0.1 |

### B.1 Network Instance Formulation

Recall, each network instance $I = (G, X, Y, \pi)$ in our framework consists of four primary components:

- A network topology $G = (V, E)$ defining the nodes and directed links in the network
- A traffic specification $\mathcal{K} = \{(dest_k, \boldsymbol{\lambda}_k)\}_{k=1}^{K}$ where $dest_k \in V$ is the destination node for class $k$ and $\boldsymbol{\lambda}_k = \{\lambda_{i,k}\}_{i \in V}$ specifies the external arrival rates of class $k$ packets at each node $i$
- A set of link service rates $\mathcal{U} = \{\mu_{i,j}\}_{(i,j) \in E}$ defining the capacity of each link in the network
- The network control policy $\pi$ which dictates the dynamics of the queueing network (i.e. how packets are buffered, what control action is taken in each time step, and scheduling constraints), and the role of the network control coefficients.

To evaluate routing performance across these instances, we developed a network emulator that, given a network instance $I$, time horizon $T$, and routing coefficient matrix $W$, produces a trajectory $\tau = (\mathbf{S}(t), \mathbf{A}(t))_{t=1,\ldots,T}$. Here, $\mathbf{S}(t)$ represents the network state at time $t$ (including queue lengths and link states), and $\mathbf{A}(t)$ captures the routing decisions made by policy $\pi$ in state $\mathbf{S}(t)$ using the routing coefficient matrix $\mathbf{W}$. From this trajectory, we compute a sample utility $\hat{U}_T(I, W; \tau)$. For both training and evaluation, the time horizon $T = 1000$ was used. Additionally, for evaluation and benchmarking, the same $\hat{U}_T(I, W)$ was estimated by taking the mean of three different runs using a new random seed to generate each trajectory.

The emulator supports both the state-independent randomized routing paradigm $\pi^{(TS)}$, the state-dependent backpressure routing paradigms without interference $\pi^{(BP)}$ and with interference $\pi^{(BPI)}$, implementing the appropriate queueing dynamics for each. This unified framework allows us to evaluate the same topology $G$, node features $X$, and link features $Y$ under different network control policies, enabling direct comparisons between state-dependent and state-independent approaches.

#### B.1.1 Random Network Generation Process

To test the generalization capabilities of our IA-GNN architecture, we developed a random network generation framework that creates diverse stochastic queueing networks based on a sequence of random sampling procedures. The generation process follows these steps:

1. **Node Generation:** Sample the number of nodes $|V|$ from a node sampling distribution
2. **Topology Generation:** Generate a network topology $G = (V, E)$ using the Barabási-Albert preferential attachment model, which creates networks with realistic properties including the small-world phenomenon and scale-free degree distribution commonly observed in communication networks.
3. **Link Rate Assignment:** For pair of bi-directional links $(i, j), (j, i) \in E$, sample service rate $\mu_{i,j} = \mu_{j,i}$ from a link rate sampling distribution. (Optional) If creating the random network for power minimization, sample link power $P_{i,j}$ from a power sampling distribution.
4. **Traffic Class Generation:**
   (a) Sample a class density parameter that determines what fraction of $|V|$ nodes will be assigned as destinations for traffic classes, thus determining $K$ for this network instance.
   (b) For each class $k \in \{1, ..., K\}$, sample an arrival node density parameter that specifies the fraction of nodes that receive external arrivals for this class.
   (c) For each external arrival rate $\lambda_{i,k}$ designated as non-zero based on the arrival node density, sample its value from an arrival rate generating distribution

This generation process creates network instances with substantial variation across all relevant dimensions: topological structure (varying node counts, edge densities, path lengths, and graph diameters), link characteristics (varying service rates), and traffic patterns (varying numbers of classes, arrival rates at each node for each class). The resulting test set provides a robust benchmark for evaluating the generalization capabilities of our proposed approach.

By training on a diverse set of randomly generated instances and testing on previously unseen instances from the same distribution, we can assess whether our MA-GNN architecture learns generalizable routing strategies that adapt effectively to new network environments without requiring retraining.

### B.2 Environment Set Statistics

Two distinct sets of environments were generated for the experiments presented in Section 6 of the main paper. Environment Set #1 was used for both training and evaluation of the models aimed at optimizing the Time-Averaged Delay and Worst-Case Class Delay performance metrics. Environment Set #2, on the other hand, was exclusively used for training under the Mean Power Minimization with Throughput Constraints performance metric, with models trained specifically under the Backpressure with Wireless Interference Policy ($\pi^{(BPI)}$).

Both environment sets were generated following the process outlined earlier, with the sampling distributions and all associated environment parameters fully documented in the codebase. In this section, we report statistical summaries of the metrics collected across all environments in each set. These statistics highlight the variability within each environment set, emphasizing the challenge faced by the trained models in generalizing to the diverse conditions they were evaluated under.

Table 2: Description of Network Metrics

| Metric | Description |
|---|---|
| *Basic Network Properties* | |
| Number of Nodes | Total count of nodes in the network |
| Number of Links | Total count of directed links connecting nodes in the network |
| Number of Classes | Total count of traffic classes, each with a distinct destination |
| *Connectivity Metrics* | |
| Average Degree | Average number of birectional links per node |
| Network Density | Ratio of actual links to potential links in the network |
| Network Diameter | Maximum shortest path length between any pair of nodes |
| *Traffic Metrics* | |
| Total Arrival Rate | Sum of all traffic arrival rates across all classes |
| Max Class Arrival Rate | Highest arrival rate among all traffic classes |
| Min Class Arrival Rate | Lowest arrival rate among all traffic classes |
| Std Class Arrival Rate | Standard deviation of arrival rates across classes |
| *Link Capacity Metrics* | |
| Average Link Rate | Mean capacity (service rate) across all links |
| Min Link Rate | Minimum capacity of any link in the network |
| Max Link Rate | Maximum capacity of any link in the network |
| Std Link Rate | Standard deviation of link capacities |
| *Path and Flow Metrics* | |
| Average Path Diversity | Average number of edge-disjoint paths between source-destination pairs |
| Min Path Diversity | Minimum number of edge-disjoint paths among all source-destination pairs |
| Average Flow Ratio | Average ratio of arrival rate to maximum possible flow rate |
| Max Flow Ratio | Maximum ratio of arrival rate to maximum possible flow rate |
| Unsatisfiable Flows | Count of flows where arrival rate exceeds maximum possible flow rate |
| *Performance Metrics* | |
| SPBP Backlog | Average queue backlog using Shortest Path Biased Backpressure routing |
| SPBP Throughput | Throughput achieved using Shortest Path Biased Backpressure routing |
| SPBP Max Class Backlog | Maximum average queue backlog among all classes using Shortest Path Biased Backpressure routing |
| SPBP Power | Network power (throughput/delay trade-off) using SPBP routing under the wireless interference model |
| SPTS Backlog | Average queue backlog using Shortest Path Traffic Splitting routing |
| SPTS Throughput | Throughput achieved using Shortest Path Traffic Splitting routing |
| SPTS Max Class Backlog | Maximum average queue backlog among all classes using Shortest Path Traffic Splitting routing |
| EECA Backlog | Average queue backlog using the Energy Efficient Control Algorithm (EECA) [1] |
| EECA Throughput | Throughput achieved using the EECA |
| EECA Power | Maximum average queue backlog among all classes using EECA |

Table 3: Environment Set #1 Statistics

| Metric | Mean | Std | Min | Max |
|---|---|---|---|---|
| Number of Nodes | 12.47 | 1.41 | 10.00 | 14.00 |
| Number of Links | 41.88 | 5.64 | 32.00 | 48.00 |
| Number of Classes | 3.12 | 0.85 | 1.00 | 5.00 |
| Average In-Degree | 3.35 | 0.08 | 3.20 | 3.43 |
| Network Density | 0.30 | 0.03 | 0.26 | 0.36 |
| Network Diameter | 3.29 | 0.52 | 2.00 | 4.00 |
| Total Arrival Rate | 8.17 | 3.08 | 2.19 | 16.67 |
| Max Class Arrival Rate | 4.05 | 1.27 | 1.90 | 7.85 |
| Min Class Arrival Rate | 1.40 | 0.94 | 0.12 | 4.82 |
| Std Class Arrival Rate | 1.14 | 0.54 | 0.00 | 2.92 |
| Average Link Rate | 2.48 | 0.26 | 1.86 | 3.12 |
| Min Link Rate | 1.00 | 0.00 | 1.00 | 1.00 |
| Max Link Rate | 4.00 | 0.00 | 4.00 | 4.00 |
| Std Link Rate | 1.09 | 0.09 | 0.78 | 1.27 |
| Average Path Diversity | 2.35 | 0.30 | 1.71 | 3.25 |
| Min Path Diversity | 1.89 | 0.31 | 1.00 | 2.00 |
| Average Flow Ratio | 0.17 | 0.05 | 0.08 | 0.35 |
| Max Flow Ratio | 0.45 | 0.15 | 0.18 | 0.78 |
| Unsatisfiable Flows | 0.00 | 0.00 | 0.00 | 0.00 |
| SPBP Backlog | 146.58 | 109.73 | 26.62 | 489.26 |
| SPBP Throughput | 0.98 | 0.01 | 0.92 | 1.00 |
| SPBP Max Class Backlog | 91.51 | 55.33 | 16.51 | 231.25 |
| SPTS Backlog | 598.78 | 447.75 | 22.67 | 1896.06 |
| SPTS Throughput | 0.85 | 0.10 | 0.58 | 0.99 |
| SPBP Max Class Backlog | 403.14 | 285.96 | 14.87 | 1377.82 |

Table 4: Environment Set #2 Statistics

| Metric | Mean | Std | Min | Max |
|---|---|---|---|---|
| Number of Nodes | 12.17 | 1.34 | 10.00 | 14.00 |
| Number of Links | 40.68 | 5.37 | 32.00 | 48.00 |
| Number of Classes | 2.80 | 0.80 | 1.00 | 5.00 |
| Average In-Degree | 3.33 | 0.08 | 3.20 | 3.43 |
| Network Density | 0.30 | 0.03 | 0.26 | 0.36 |
| Network Diameter | 3.24 | 0.43 | 3.00 | 4.00 |
| Total Arrival Rate | 1.83 | 1.20 | 0.23 | 5.64 |
| Max Class Arrival Rate | 1.17 | 0.79 | 0.19 | 4.13 |
| Min Class Arrival Rate | 0.25 | 0.31 | 0.00 | 1.51 |
| Std Class Arrival Rate | 0.41 | 0.33 | 0.00 | 1.57 |
| Average Link Rate | 4.99 | 0.32 | 4.17 | 5.67 |
| Min Link Rate | 3.01 | 0.10 | 3.00 | 4.00 |
| Max Link Rate | 7.00 | 0.00 | 7.00 | 7.00 |
| Std Link Rate | 1.38 | 0.14 | 0.91 | 1.75 |
| Average Path Diversity | 2.37 | 0.42 | 1.00 | 3.50 |
| Min Path Diversity | 1.81 | 0.39 | 1.00 | 2.00 |
| Average Flow Ratio | 0.03 | 0.02 | 0.01 | 0.09 |
| Max Flow Ratio | 0.09 | 0.08 | 0.01 | 0.48 |
| Unsatisfiable Flows | 0.00 | 0.00 | 0.00 | 0.00 |
| EECA Backlog | 72.92 | 46.01 | 11.33 | 228.42 |
| EECA Throughput | 0.97 | 0.01 | 0.93 | 1.00 |
| EECA Power | 10.88 | 3.27 | 3.46 | 19.56 |

## C Architecture Details

Our proposed Multi-Axis GNN (MA-GNN) model builds upon a deep graph neural network architecture tailored to capture both inter-node and intra-node message passing dynamics in multi-class queueing networks. The network is constructed as a deep stack of modified message-passing layers, each augmented with intra-node aggregation to enable information sharing across traffic classes at each node.

To ensure effective training at depth, we adopt the normalization and residual connection strategy introduced in [2] known to significantly increase the maximum trainable depth of GNNs. Incorporating these architectural enhancements was essential for our setting, as it was critical that the GNN's receptive field exceed the diameter of each graph seen during training and evaluation.

The number of GNN layers determines how far node and edge feature information can propagate. In our application, a particularly important input feature is the destination indicator for each traffic class—this feature is only active at the destination node. If the distance between a given node and the destination for a class exceeds the receptive field of the GNN, the model will be unable to incorporate that information, and thus unable to make informed routing decisions for packets arriving at that node.

To overcome this, we leverage the depth-scalability improvements introduced by the DeeperGCNs framework, enabling us to train GNNs with sufficient depth to fully cover the diameter of all graphs encountered in our experiments. All models trained in this work had a depth of $L = 10$ and a hidden dimension of $F_\ell = 32$ for all layers.

The layers of the MA-GNN used in all experiments were based around using concatenation and MLPs to combine the input embeddings for message construction, node embedding updates, and edge embedding updates. Thus the operations of each GNN layer can be expressed as:

$$\mathbf{M}_{j,i}^{(\ell)} = \rho^{(\ell)}(\mathbf{H}_j^{(\ell)}, \mathbf{H}_{i,j}^{(\ell)}) = \text{MLP}^{(\ell)}(\text{concat}(\mathbf{H}_j^{(\ell)}, \mathbf{H}_{i,j}^{(\ell)})) \quad \text{(Internode Message Construction)}$$

$$\mathbf{M}_i^{(\ell)} = \psi^{(\ell)}\left(\{\mathbf{M}_{j,i}^{(\ell)} \mid j \in \mathcal{N}(i)\}\right) = \text{AGG}\left(\{\mathbf{M}_{j,i}^{(\ell)} : j \in \mathcal{N}(i)\}\right) \quad \text{(Internode Message Aggregation)}$$

$$\mathbf{V}_i^{(\ell)} = \omega^{(\ell)}(\mathbf{H}_i^{(\ell)}) = \text{MLP}(\mathbf{H}_i^{(\ell)}) \quad \text{(Intranode Message Construction)}$$

$$\mathbf{n}_{i,k}^{(\ell)} = \Gamma^{(\ell)}(\{\mathbf{v}_{i,k'}^{(\ell)} : k' \in 1, ..., K\}) = \text{AGG}(\{\mathbf{v}_{i,k'}^{(\ell)} : k' \in 1, ..., K\}) \Rightarrow \mathbf{N}_i \quad \text{(Intranode Message Aggregation)}$$

$$\mathbf{H}_i^{(\ell+1)} = \phi^{(\ell)}\left(\mathbf{H}_i^{(\ell)}, \mathbf{M}_i^{(\ell)}, \mathbf{N}_i^{(\ell)}\right) = \text{MLP}(\text{concat}(\mathbf{H}_i^{(\ell)}, \mathbf{M}_i^{(\ell)}, \mathbf{N}_i^{(\ell)})) \quad \text{(Node Embedding Update)}$$

$$\mathbf{H}_{i,j}^{(\ell+1)} = \eta^{(\ell)}(\mathbf{H}_i^{(\ell+1)}, \mathbf{H}_j^{(\ell+1)}, \mathbf{H}_{i,j}^{(\ell)}) = \text{MLP}(\mathbf{H}_i^{(\ell+1)}, \mathbf{H}_j^{(\ell+1)}, \mathbf{H}_{i,j}^{(\ell)}) \quad \text{(Edge Embedding Update)}$$

where $\text{AGG}(\cdot)$ is a permutation invariant set function such as an element-wise sum, mean, or scaled dot product attention (SDPA). We found that simple concatenation followed by an MLP yielded strong empirical performance, likely due to its flexibility in learning nonlinear interactions between inputs. In contrast, using summations to combine embeddings e.g. $\phi^{(\ell)}\left(\mathbf{H}_i^{(\ell)}, \mathbf{M}_i^{(\ell)}, \mathbf{N}_i^{(\ell)}\right) = \sigma(\mathbf{H}_i^{(\ell)}\mathbf{\Theta}_{\phi_1}^{(\ell)} + \mathbf{M}_i^{(\ell)}\mathbf{\Theta}_{\phi_2}^{(\ell)} + \mathbf{N}_i^{(\ell)}\mathbf{\Theta}_{\phi_3}^{(\ell)})$ where $\sigma(\cdot)$ is a non-linear activation function

and $\{\mathbf{\Theta}_{\phi_n}\}_{n=1,..,3}$ are learnable weight matrices, did not yield strong performance. The architecture implementation can be found in the codebase.

## References


[1] M.J. Neely, E. Modiano, and C.E. Rohrs. Dynamic power allocation and routing for time-varying wireless networks. *IEEE Journal on Selected Areas in Communications*, 23(1):89–103, January 2005. Conference Name: IEEE Journal on Selected Areas in Communications.

[2] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. DeeperGCN: All You Need to Train Deeper GCNs, June 2020. arXiv:2006.07739 [cs].