Signal Processing on Graphs and Networks Final Project

Differentiable Digital Twins in Graph Neural Networks

Elad Sofer 312124662
Tomer Shaked 315822221

Abstract

In Graph Signal Processing research, the static nature of the adjacency matrix limits its adaptability to dynamic real-world systems. Recognizing the potential of dynamic graphs, where the adjacency matrix can be dynamically altered, we delve into optimizing such systems for specific objectives, such as minimizing packet loss in communication networks. We explore two methodologies: leveraging differentiable simulators and employing Graph Neural Networks (GNNs) as digital twins. Our study showcases that while differentiable simulators offer optimal performance, GNN-based digital twins provide a viable alternative, albeit with slightly lower efficacy. Through extensive experimentation, employing datasets generated from Erdos Renyi graphs, we demonstrate the efficiency of our approaches, shedding light on their strengths and areas for improvement. Our findings underscore the potential of differentiable digital twins in enhancing the adaptability and optimization of dynamic graphs in real-world scenarios.

1. Introduction

In most Graph Signal Processing research the adjacency matrix is a static input which is only used as a mean to help process the signal itself. Though in reality many systems in the real world have connections which could be parameterized. For example, in a computer networking scheme, connections could be added or improved between routers, In a power grid, power lines can be reinforced, and in road engineering one could change the road quality or change the amount of lanes to change the capacity of the road [1]. Usually, these network configurations are bound by some constraint usually relating to financial limitations. We thus name graphs where the adjacency matrix can be modified and altered as programmable graphs. The programmability of these graphs enables optimizing their systems and adapting to dynamic variations based on a desired property, e.g., maximizing throughput on the graph [2].

When operating in complex settings, one is expected to be able to relate the tunable parameters to the resulting environment only via a complex simulator. The absence of a simple closed-form expression relating the environment to the tunable parameters notably complicates their optimization. Existing model-free approaches for non-graph optimization are based on Bayesian optimization [3] or reinforcement learning [4], either struggle in tuning many parameters, or tend to be extremely lengthy.

An emerging approach to model complex wireless channels is based on digital twins. The term digital twin refers to a software replica of a physical environment [5]. Digital twins accommodate both traditional simulators, as well as DNN-based models. The latter has been the focus of growing interest in the context of generative models, i.e., to generate and evaluate graphs [6]. Nonetheless, a key property of many digital twins (and DNN-based digital twins in particular), is their end-to-end differentiability, that makes them amenable to efficient optimization based on first-order methods. This motivates exploring such differentiable digital twins for surrogate model-based optimization of programmable graphs.

In this work we study the optimization of programmable graphs with first-order optimization methods for a specific problem of packet loss minimization on a network. We explore two techniques to do so: The first approach relies on the availability of a simulator that is both accurate and differentiable, and uses it as a differentiable digital twin for first-order optimization with a relatively small number of iterations. While this approach is restricted to settings where a differentiable simulator exists, we introduce a second approach, where we show that such digital twins can also be learned from measurements using GNNs. Our experiments show that optimizing via differentiable twins facilitates rapid graph parameters tuning, and that while differentiable simulators provide the best performance, they can be approximated using DNNs. The latter also supports rapid optimization, while being operable without requiring faithful software simulators.

2. Related Works

We've seen a few works which have used digital twins in graph processing, but these have not been in use in conjunction with first order methods through the model or have not attempted to optimize the adjacency matrix directly. For example, in [7] and [6] the papers built a digital twin to model the network but they have used it as a forward model to choose the optimized configuration. In [8] the authors optimized the adjacency matrix using reinforcement learning and in [9] the adjacency matrix is optimized according to a few chosen parameters and a crafted cost function, This method offers the engineers much more control but could result in a suboptimal result.

3. System Model

We imagine a communication network system which exhibits packet loss between each two nodes in the network. X is the Traffic matrix and each row X_i is the traffic vector of node i, while $X_{i,j}$ is the packet amount destined from node i to node j in the system.

Moreover, the adjacency matrix A serves as a matrix indicating delivery probabilities. Within the network, each node can send it's packets via any transmission route. However, when packets traverse a connection (edge) $\{m, n\}$, there exists a probability of $1 - A_{m,n}$ for them to be lost. Thus, meticulous path selection is imperative to mitigate packet loss (a goal which can be achieved through the utilization of graph algorithms such as Bellman Ford and Floyd Warshall).

We consider a graph $G(V, E)$ along with an adjacency matrix A. As mentioned previously, the entry $A(i, j) = A_{i,j}$ signifies the likelihood of a packet successful transmission (without a loss) from a source node i to a destination node j. Specifically, for a given source node i and destination node j, we denote $\{T^{ij}\}^M_{l=1}$, the set of all paths from node i to node j where:

$$path\ k\colon\ T_k^{ij} = \{v_i = v_{k_1}, v_{k_2}, \dots, v_{k_{N-1}}, v_j = v_{k_N}\}$$

Since a successful transmission upon each edge is independent to a different edge successful transmission, then the probability for a packet to pass through path k is:

$$delivery\ probabilty\ for\ path\ k\ from\ i\ to\ j\colon p_k^{ij} = \prod_{n=1}^{N-1} A\,(k_{n+1}, k_n)$$

We assume that each node sends packets in the optimal path so the probability for a packet to get from it's source i to it's destination j is:

$$delivery\ probabilty\ from\ i\ to\ j: p^{ij} = \max_{k;T_k^{ij}} p_k^{ij}$$

Hence, the amount of packets delivered from the source node i to destination node j is given by:

$$delivery\ from\ i\ to\ j: d^{ij} = p^{ij} * x^{ij}$$

Where x^{ij} is the amount of packets sent from source node i to destination node j.
We can denote d, the total deliveries in the system and it is equal to:

$$total\ deliveries : d = \sum_{i\in V} \sum_{j\in V, j\neq i} d^{ij} = \sum_{i\in V}\sum_{j\in V} p^{ij}x^{ij} = 1^T \cdot P^T X \cdot 1$$

We can substitute p^{ij} in the last equation with it's definition to get the final equation:

$$d = \sum_{i\in V}\sum_{j\in V} x^{ij} \max_{k;T_k^{ij}} \prod_{n=1}^{N-1} A\left(k_{n+1}, k_n\right) \qquad (1)$$

Given that matrix A remains constant, this problem can be effectively solved numerically by utilizing a search algorithm to identify the optimal route between each source and destination.

2.1.Problem formulation

Now, introducing an additional complexity: the adjacency matrix A, representing the probability of package drops, can be adjusted to maximize delivery rates, provided that the following constraint is maintained:

$$\sum_{ij} A_{ij} \leq P * |E| \quad (2)$$

Given that P is the average probability of successful transmission across an edge, and $|E|$ denotes the total number of edges in the graph. This constraint can be thought of as a financial limit where we have a budget $P * |E|$, and we need to choose where to spend it on. We will note the subspace of adjacency matrixes which are subject to (2) and their elements are limited between 0 and 1 as such:

$$\mathcal{P} = \left\{A \middle| 0 < A_{i,j} < 1, \sum_{ij} A_{ij} \leq P * |E|\right\}$$

The central inquiry guiding this project becomes: **How might we determine the ideal adjustment of graph weights to maximize the deliveries**?

$$\hat{A} = \underset{A\in\mathcal{P}}{argmax}(d) = \underset{A\in\mathcal{P}}{argmax}\left(\sum_{i\in V}\sum_{j\in V} x^{ij} \max_{k;T_k^{ij}} \prod_{n=1}^{N-1} A\left(k_{n+1}, k_n\right)\right) \quad (3)$$

This task is challenging due to the fact that modifying any weight in the adjacency matrix impacts multiple paths, potentially altering the optimal route between two vertices and significantly impacting the deliveries.

3. Proposed Solution

3.1. Optimization on differentiable digital twins

A candidate approach to tackle (3) employs gradient-based optimization. Notice that the function in (1) given static pathing's T is differentiable. Which Enables us to compute:

$$\nabla_A[d_T] = \sum_{i\in V}\sum_{j\in V}\nabla_A\left[{d_T}^{i,j}\right]$$

Where d_T symbolizes the deliveries on pathing's T. Then, (3) can be tackled via PGA iterations of the form

$$A^{(k+1)} \leftarrow \Pi_{\mathcal{P}}\left(A^{(k)} + \mu\nabla_A[d_T]\right) \quad (4)$$

where k is the iteration index, $\mu > 0$ is the step-size, and $\Pi_{\mathcal{P}}(\cdot)$ denotes (elementwise) projection onto the feasible set $\mathcal{P}$. We will Suggest two types of digital twins that can solve this problem, a simulation based digital twin which works by using a greedy path choice and a GNN based digital twin which simulates the deliveries with a deep neural network.

3.2. Simulation based Digital Twin

The first option is to limit the optimization to a greedy optimization. If we split our optimization to the questions of choosing optimal pathing's and choosing the best weight matrix for a pathing, we then get two problems which could be solved much more easily. Our simulator algorithm chooses the best path between each source and destination assuming a fixed adjacency matrix just as in (1). We then freeze the paths and use them to generate a cost function which we can use to optimize A as in equation (4). We go back and forth between the best-path estimation process (Floyd-Warshall) and the edge optimization process until we reach convergence.

ALGORITHM 1: VECTORIZED FLOYD-WARSHALL

```
   Input: Adjacency matrix $A$
   Output: distance Matrix $\mathcal{D}$, pathing's matrix $\mathcal{R}$
1  Initialize: $\mathcal{D} \leftarrow A$;  $\mathcal{R}_{i,j} = i, \forall i, \forall j$
2  Initialize: Alternative Pathing's Matrix $Q$ initialized to zeros.
3  for k=0, 1,…, |V| do
4     Calculate $\boldsymbol{Q} \leftarrow \boldsymbol{\mathcal{D}}[:,\boldsymbol{k}] * \boldsymbol{\mathcal{D}}[\boldsymbol{k},:]$   // Best pathing's for subgraph 0,…,k
5     Calculate $\boldsymbol{\mathcal{D}} \leftarrow maximum\ between\ \boldsymbol{\mathcal{D}}\ and\ \boldsymbol{Q}$
6     Calculate $\boldsymbol{\mathcal{R}_{i,j}} \leftarrow \boldsymbol{\mathcal{R}_{k,j}}\ for\ \boldsymbol{\mathcal{D}_{ij}} \le \boldsymbol{Q_{ij}}$ else $\boldsymbol{\mathcal{R}_{i,j}}$
7  End
```

Path Finding

Finding the best path between sources and destinations on a weighted graph is a solved problem and has a few solutions. The classic Dijkstra algorithm [10] solves this problem for a single source and destination (though one could simply repeat it to find all paths between source and destination) and has a worst case performance of $O\ (|E||V|^2 + |V|^3\ log|V|)$. The Bellman ford algorithm works a bit more efficiently and can also support negative edges (which we don't care about) with a performance of $O\ (|E||V|^3)$. Though since we want the distance between every node we chose to use the Floyd-Warshall algorithm [11][1] which solves the shortest paths between every source and destination at once and has the worst case performance of $O\ (|V|^3)$. To fully utilize our resources, we implemented a vectorized version

[1] Interestingly This algorithm seems to be published inside a book of algorithms and it is simply mentioned as 'algorithm 97' showing that sometimes you really can't predict what algorithms will be a success. Also, apparently Bernard Roy already published this algorithm 3 years prior but did not get the same recognition for it.

of this algorithm which uses the adjacency matrix as an initial step. The algorithm is summarized in Alg. (1). From this algorithm matrix $\mathcal{R}$ can then be retraced from end to beginning to generate the paths.

Edges Optimization

Given specific chosen paths T from step 1 we now have a closed form differentiable equation to calculate the deliveries:

$$d_T(A) = \left(\sum_{i\in V} \sum_{j\in V} x^{ij} \prod_{\substack{n=1 \\ T_k^{ij}\, from \text{ path finding}}}^{N-1} A\left(k_{n+1}, k_n\right) \right)$$

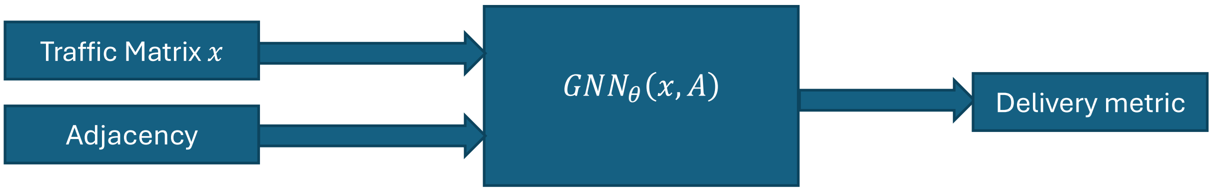
Using Projected gradient ascent as in equation (4) we can then optimize the Adjacency matrix.

3.3. GNN based Digital Twin

We propose the training of a Graph Neural Network (GNN) comprising of 3 Graph Convolution Layers (GCN), culminating in two fully connected layers, for the purpose of estimating the delivery of a network comprised of a traffic matrix X and adjacency matrix A.

The GNN's output $\hat{d}$ is denoted as:

$$\hat{d} = GNN_\theta(x, A) = G_\theta(x, A)$$

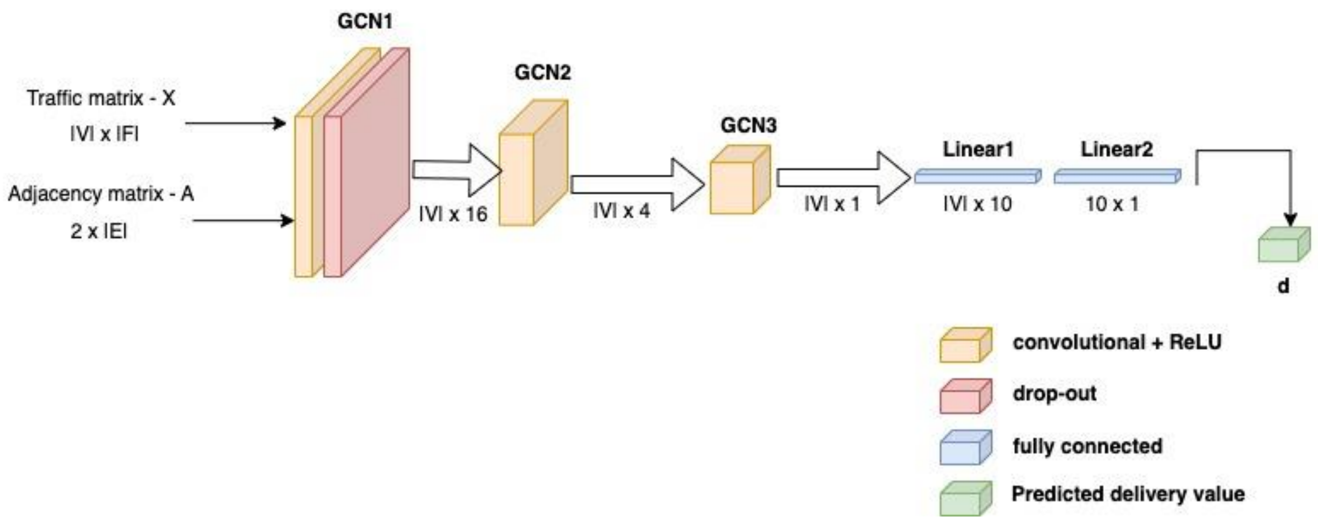


Specifically, we employ the graph convolutional network layer taken from "Semi-supervised Classification with Graph Convolutional Networks" paper. [12]

where the $l + 1$ hidden layer is defined as:

$$h^{l+1} = D^{-1/2} A D^{-1/2} \mathrm{h}^l \theta^l$$

While D is the diagonal degree matrix, A is the adjacency matrix and θ is the $l'th$ Graph Convolution Layer (GCN) parameters. Moreover, to avoid over-fitting we used a dropout giving the following overall architecture:



Training this GNN yields a differentiable digital twin representing the network.
After we have a trained digital twin of the network, we utilize gradient ascent-based methods to optimize the network's adjacency matrix:

$$A_{n+1} = \Pi_{\mathcal{P}}(A_n + \mu \nabla_A G_\theta(x, A))$$

Here, μ signifies the learning rate of the optimization process, and $\Pi_{\mathcal{P}}$ is the projection function defined as:

$$\Pi_{\mathcal{P}}(x)_i = \frac{|x_i|}{\|x\|_1} \cdot P \cdot |E|$$

4. Experimental Results

The dataset is comprised of a single value of input x for a Erdos Renyi graph G with 25 vertices and edge probability of 15%. The adjacency matrix weights A are different for each entry in the dataset. We set the average delivery probability on edges as P=0.6. For each graph and adjacency matrix we calculated the optimal pathing using Bellman Ford and then calculated the deliveries using equation (1). The Generated dataset has the following structure $D = \{\tilde{x}, A_i, d_i\}_{i=1}^{N}$.
We trained the GNN on a dataset of N=30,000 samples of such graphs for 500 epoches. The training process loss curve can be viewed in figure (1)

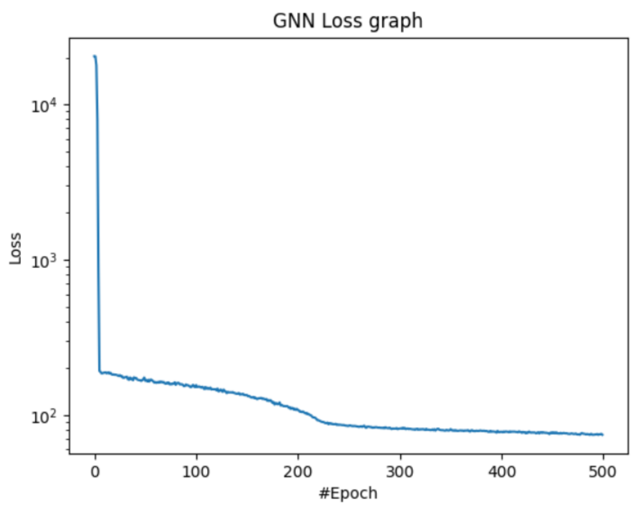


Figure 1: Loss curve for GNN training.

We can compare the optimization process on the greedy simulator based digital twin to the GNN based digital twin in Fig. (2). We can see that both algorithms reached an improvement in the deliveries for two different initial graphs. the simulator based digital twin produce better results overall compared to the GNN digital twin though the GNN had much smaller latency(2s for GNN optimization and 1m 39s for the .

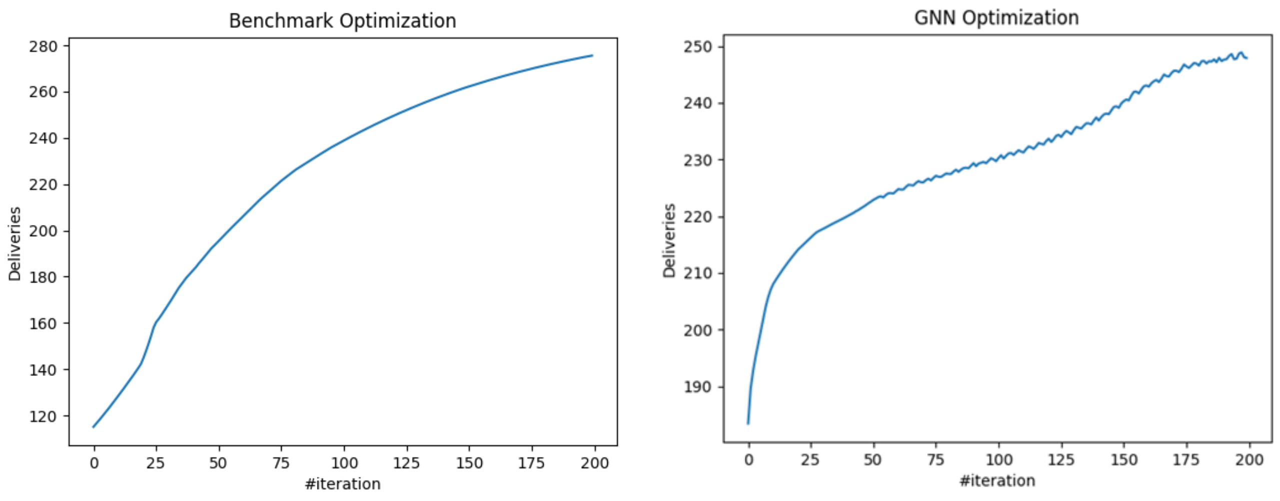


Figure 2: Benchmark Optimization Process for the benchmark (Simulation) on the left, and for the GNN on the right. The two algorithms start from a different initial graph which causes the difference in the initial point.

We can see in figure (3) visually how using the optimization process creates better pathing's on the graph. Notice how before the optimization the weights were scattered around the graph and after the optimization we have a few main paths running along the graph which have a point of contact with almost all nodes on the graph.

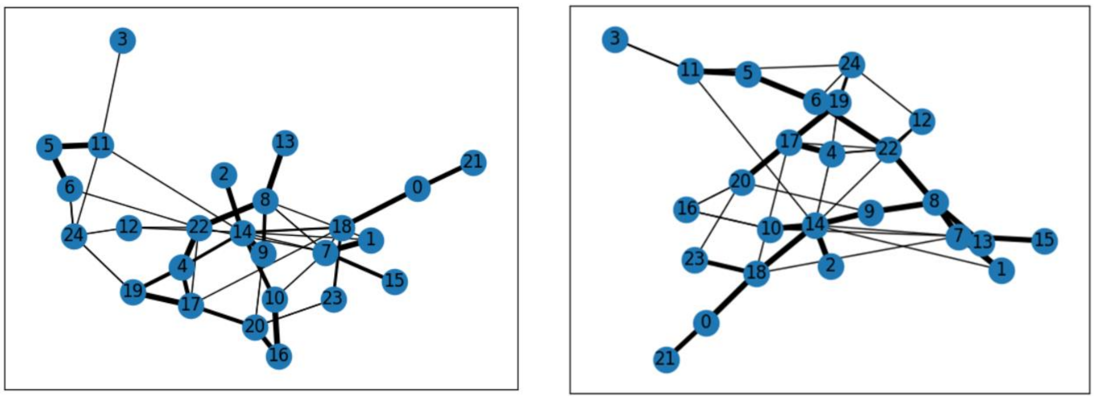


Figure 3: two images of a network before(left) and after(right) optimization. Edge width corresponds with the weight of the edge. we use the simulator optimization in this figure because it has the overall better performance.

5. Discussion

The greedy algorithm advantages are it's accuracy and performance. Compared to it we get a bit worse results for the GNN model but the upside of the GNN model is it's faster computation which might be critical in real-world scenarios and it being model-free which allows us to use this model in more complicated scenarios where simulators are not available.

6. Bibliography

[1] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE,* vol. 106, p. 808–828, 2018.

[2] A. Brzezinski, G. Zussman and E. Modiano, "Enabling distributed throughput maximization in wireless mesh networks: a partitioning approach," in *Proceedings of the 12th annual international conference on Mobile computing and networking*, 2006.

[3] L. Wang, N. Shlezinger, G. C. Alexandropoulos, H. Zhang, B. Wang and Y. C. Eldar, "Jointly learned symbol detection and signal reflection in RIS-aided multi-user MIMO systems," in *2021 55th Asilomar Conference on Signals, Systems, and Computers*, 2021.

[4] K. M. Faisal and W. Choi, "Machine learning approaches for reconfigurable intelligent surfaces: A survey," *IEEE Access,* vol. 10, p. 27343–27367, 2022.

[5] Y. Wang, Z. Su, S. Guo, M. Dai, T. H. Luan and Y. Liu, "A survey on digital twins: architecture, enabling technologies, security and privacy, and future prospects," *IEEE Internet of Things Journal,* 2023.

[6] B. Li, T. Efimov, A. Kumar, J. Cortes, G. Verma, A. Swami and S. Segarra, "Learnable Digital Twin for Efficient Wireless Network Evaluation," in *MILCOM 2023-2023 IEEE Military Communications Conference (MILCOM)*, 2023.

[7] M. Ferriol-Galmés, J. Suárez-Varela, J. Paillissé, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros and A. Cabellos-Aparicio, "Building a Digital Twin for network optimization using Graph Neural Networks," *Computer Networks,* vol. 217, p. 109329, 2022.

[8] Y. Zhang, H. Ren, J. Ye, X. Gao, Y. Wang, K. Ye and C.-Z. Xu, "Aoam: Automatic optimization of adjacency matrix for graph convolutional network," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021.

[9] C. Mannweiler, P. Chakraborty and H. Schotten, "Multi-objective adjacency matrix optimization for coordinated wireless backhaul networks," in *2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2013.

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, 2022, p. 287–290.

[11] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM,* vol. 5, p. 345–345, 1962.

[12] T. N. K. a. M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *CoRR*, 2016.

[13] M. R. Izadi, Y. Fang, R. Stevenson and L. Lin, "Optimization of graph neural networks with natural gradient descent," in *2020 IEEE international conference on big data (big data)*, 2020.