UNIVERSITÄT BONN

Fraunhofer FKIE

Master Thesis:
Link Prediction in Dynamic Communication Networks Using Graph Neural Networks

Project Summary

Problem Statement

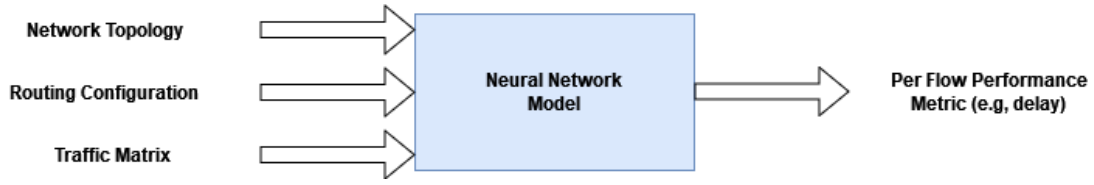


Figure 1: Schematic representation of a neural network based communications network model

Given a communication network state (topology, routing configuration, and traffic), can the network's performance be measured using a Graph Neural Network (GNN) model? The proposed solution is trained and evaluated on a dataset generated using the OMNet++ packet-level network simulator, the final accuracy is measured using the Mean Absolute Percentage Error (MAPE) metric computed on all the paths of the network.

Overview

Disconnected, Intermittent and Limited (DIL) communication networks occur in various mission-critical applications (e.g. search and rescue in remote areas). These networks are highly dynamic, and constrained by their environment; however, like other communication networks they can be naturally represented as a set of connected nodes and edges. For optimizing and improving a communication network, building a model that can predict the development of the network connections and the quality of the links would be very beneficial. This model should learn a representation that is capable of generalizing to unseen networks. Ideally, this model should also scale to model larger communication networks. Traditional neural network architectures like CNNs and RNNs are generally ill-suited to learn meaningful representations from non-euclidean data structures like graphs. GNNs are a novel approach to embed dynamic, complex, and large networks into a representation that is feasible for neural networks at scale. GNNs are able to extract the underlying relationships between elements in graph-structured data by convoluting over the graph structure itself, allowing the model to generalize to unseen graphs. The focus of this work is on using GNNs for modeling communications networks and predicting their performance metrics. The communication network is represented as a heterogenous graph with different types of information attached to the nodes and edges.

Project Description

Introduction

Disconnected, Intermittent and Limited (DIL) communication networks occur in various mission-critical applications, such as search and rescue operations in remote areas. These networks are dynamic, and highly influenced by their environments, which makes them hard to encode, predict, and control. However, network modeling is essential for optimizing and finding the optimal configuration for a network. Network models should predict the performance of a network given the network configuration, this can then be combined with an optimization algorithm (e.g. Reinforcement Learning) to produce an optimal configuration given a target policy (e.g. minimize the delay). Additionally, the model should be able to generalize to larger unseen networks, which allows for training the model in a controlled testbed (e.g. in a lab) and then directly deploying it in real-world networks, without the need to retrain the model.

Traditional network modeling techniques are divided into analytic models and packet-level network simulators [1]. Analytic models, which are mainly based on queuing theory [2], hold strong assumptions that may not hold in complex networks (e.g. traffic follows a Poisson distribution); as a result, they are not very accurate at modeling large networks with realistic configurations [3]. In contrast, packet-level simulators are very accurate when modeling large scale networks; however, they are computationally very expensive and deploying them in a short time scales is often infeasible [1]. In this context, Machine Learning(ML) algorithms appear as a promising solution to build accurate, real-time models for communication networks. ML algorithms use historical data to constantly learn how to make more accurate predictions. For communications networks, graphs are a common mathematical formalism that can be used to represent the data. Graph representations allow for describing the properties of different devices using nodes in the graph and the connections between them as edges; this formalism allows for an adequate description of the topology of the network.

Standard deep learning techniques such as Convolutional Neural Networks(CNNs) or Recurrent Neural Networks (RNNs) are generally ill-suited for learning from graphs. These methods lack the flexibility of capturing the underlying sub-structure of a graph needed to generalize to arbitrary graph structures, the reason for this is that these methods are designed for euclidean data structures (e.g. sequences and grids); however, graphs are a form of non-euclidean data structures. Non-euclidean data structures have no common system of coordinates, so basic operations like resizing or shifting are not well defined. Additionally, non-euclidean data have no canonical ordering and can be arbitrarily permuted, so representing them as a flat euclidean data structure (e.g. adjacency matrix) is ambiguous. For standard deep learning techniques ambiguity in the input makes generalization difficult. Therefore, operations that are commonly used in the euclidean domain are not directly transferable to the non-euclidean domain. [4]

Graph Neural Networks (GNNs) have been proposed to mitigate these limitations and enable the operations on graph structures directly. Popularized and promoted by Google DeepMind [5], GNNs are a neural networks family designed to understand, learn, and model graph-structured data, they extract the under lying relationships between elements of a graph-structured data by convoluting over the graph structure itself. Graph Convolution enables the model to derive information and find relationships between different nodes in the graph, even in dynamic networks with variable size of nodes; consequently, this allows the model to generalization to unseen graphs. Graph Convolutions can be seen as an extension of convolutions to graph data, it gathers the current neighbouring node information and aggregates them with the previous node embedding to get a new embedding of

the node features. This can also be seen a form of graph diffusion, where the node information is spread out throughout the neighbourhood. The number of layers in a GNN defines the number of neighbourhood hops taken into consideration for computing the node embedding.

GNNs attempt to learn a suitable representation using the node and edge features of the graph; this representation contains structural and feature information about the local neighbourhood of each node. Within a GNN there are multiple graph convolution layers, these are the core building blocks of GNNs and are responsible for combining the node and edge information from the neighbouring nodes into the representation of each node. These representations are then passed through a readout layer to produce the final output. The final output can be an estimate of a node feature, a link feature or a feature of the entire graph.

Multiple GNN architectures have been proposed in the time of this writing. Most of which fall in the category of Message Passing Neural Networks (MPNNs) [6]. Although, other architectures exist [7] [8] [9], in this work we focus on MPNN for their ease of implementation and proven performance over various tasks [6]. The MPNN framework assumes that node information can be embedded in fixed-dimension vectors. In each layer the nodes are updated based on the embeddings of all the neighbouring nodes. After k layers, each embedding captures both structural and feature-based information from its k-hop neighbourhood. Formally, the k-th layer of a MPNN is formulated as follows:

$$h_u^{(k)} = UPDATE^{(k)}(h_u^{(k-1)}, AGGREGATE^{(k)}(h_v^{(k-1)}, \forall v \in N(u))) \tag{1}$$

where $h_u^{(k)}$ is the node embedding of the node u at the k-th layer and $N(u)$ is the set of all neighbouring nodes of u. The $AGGREGATE$ function uses the state of all direct neighbours v of a node u and aggregates them in a specific way. The $UPDATE$ operation uses the current state at layer k and combines it with the aggregated neighbourhood state. Different architectures differ in how they define the $AGGREGATE$ and $UPDATE$ functions. The initial embedding at $k = 0$ are the node features of each node $h_u^{(0)} = v_u, \forall v_u \in V$. An illustration of a message passing update is shown in fig.2

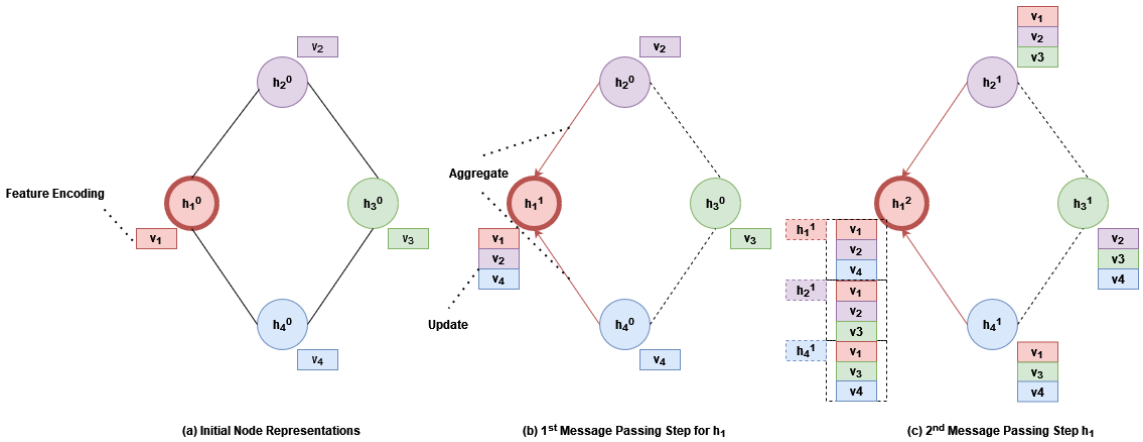


Figure 2: An illustration of the message passing update in a GNN: (a) shows the initial node features, (b) shows the intermediate representation of node h_1 after one message passing step, (c) shows the node representation for h_1 after the second message passing step for h_1

Out of all MPNN, the most expressive architecture in learning to represent and distinguish between different graph structures is the Graph Isomorphism Network (GIN) [10]. It is as expressive as the 1-Weisfeiler-Lehman (1-WL) graph isomorphism test [11], a powerful method known to distinguish a range of different graphs [12]. The WL test has been both theoretically and empirically

proven to distinguish most of the real world graphs [13]. However, in contrast to the WL test, GNNs are able to capture the similarity between different graph structures, this has been shown to be helpful for generalization [14]. Generally, there may exist other GNNs as powerful as GIN; however, GIN is one example of a maximally powerful and simple to implement GNN that follows the MPNN architecture.

GIN uses a multi-layer perceptron (MLP) for both the $AGGREGATE$ and the $UPDATE$ functions. MLPs are capable of approximating any function [15]; if they are able to approximate an injective function that is able to map different local neighborhoods to different representations, the GNN will be maximally expressive and will be able to differentiate between nodes with different local neighbourhoods. In practice, since MLPs are able to represent the composition of functions only one MLP is used to model the $UPDATE$ function and a summation is used for the $AGGREGATE$ function. A GIN update for the node u in the k-th layer is represented as

$$h_u^{(k)} = MLP^{(k)}((1-\epsilon^{(k)}).h_u^{(k-1)} + \sum_{\forall v \in N(u)} h_v^{(k-1)}) \tag{2}$$

where $\epsilon \in [0, 1]$ is a learnable parameter or a fixed scalar.

MPNNs represent a powerful learning paradigm, however, it has been shown that the expressive power of existing MPNNs are upper-bounded by the (1-WL) test [10]. This is because a MPNN will not be able to differentiate between two nodes that have a different local neighbourhood structures but the same computational graph. Identity-aware Graph Neural Networks (ID-GNNs) [16] propose a MPNN with greater expressive power than the 1-WL test. ID-GNNs are a general and powerful extension that can be applied to any MPNN, by inductively considering node identity during the embedding computation of each node, a consistent performance gain is noticed, as symmetries are broken and the number of cycles in a node are identified. An example of one failure case for MPNN is shown in fig.3 , in this example we assume that all node features are the same, we see that even though (A) and (B) have different graph structures their computational graphs are the same using MPNN, however, using ID-MPNN the two graphs produce different computational graphs.

A GNN model consists of multiple components. The overall design space consists of three sub-design spaces, the intra-layer design space which specifies the design space for each GNN layer, the inter-layer design space which states how the layers are organized, and the learning configuration which defines the training hyper-parameters for training the model. For a single layer there are four intra-layer design choices, 1) the choice of GNN aggregation and update, 2) whether to add batch normalization [17] after each layer, 3) whether to add a dropout layer [18], 4) the choice of activation function [19]. For the inter-layer design choices there are four design decisions, 1) the choice of pre-processing method, 2) whether to add skip connections, 3) the number of message passing layers, 4) the choice of post-processing method. In MPNN skip connections can be implemented as a form of Jumping Knowledge networks [20], which can flexibly leverage for each node a different neighborhood range. For the learning configurations there are also four hyperparameters, 1) the batch size, 2) the learning rate, 3) the optimizer, 4) the number of training epochs. A summary of the design space is shown in fig.4

A large set of real-world graphs are heterogeneous by nature (e.g. social graphs, communication networks). Heterogeneous graphs have different types of nodes and edges in a single graph. Learning from heterogeneous graph is notoriously challenging [21]. Input feature distributions across different nodes types are non-identical, leading to varying feature dimensionality across nodes. Additionally, heterogeneous graph learning requires learning node and edge type dependant representations, making sharing weights across different node and edge types not possible. A possible solution for dealing with heterogeneous graphs is reformulating the message passing step such that there is an

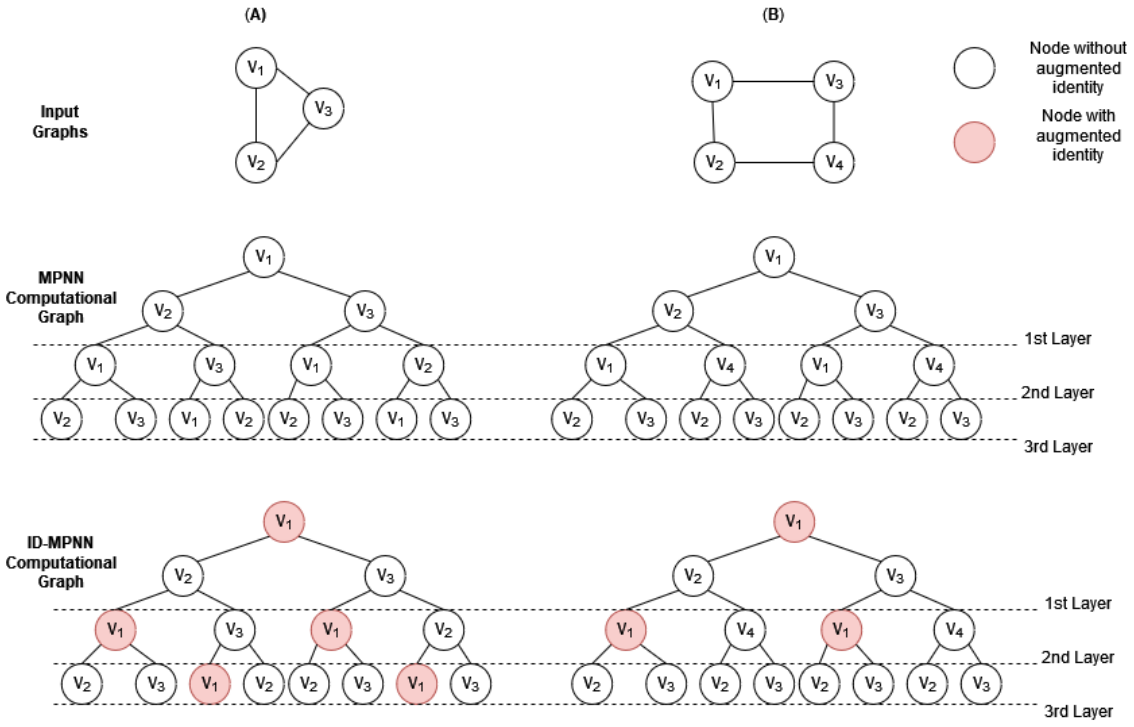


Figure 3: **Failure case for MPNN:** MPNN assign the same embedding to both nodes V_1 and V_2 because the computational graphs are identical. In contrast, ID-GNN are able to differentiation between the nodes V_1 and V_2 by inductively coloring the root node.

additional GNN layer (one which represents the message passing flow as a bipartite graph) for each edge connecting different types of nodes, and then summing them with the message passing from the nodes of the same type. An update for the node u in the k-th layer of a heterogeneous GNN is defined as:

$$h_u^{(k)} = \sum_{r \in R} GNN_\theta^{(r)}(h_i^{(l)}, h_j^{(l)} : j \in N^{(r)}(i)) \tag{3}$$

where R is the number of relations connecting nodes of different types, and each relation has its own custom GNN that uses its own relational-wise neighbourhood $N^{(r)}$. A comparison between two homogeneous GNN models and their equivalent heterogenous model if undirected edges are used to connected the different types of nodes is shown in fig.5.

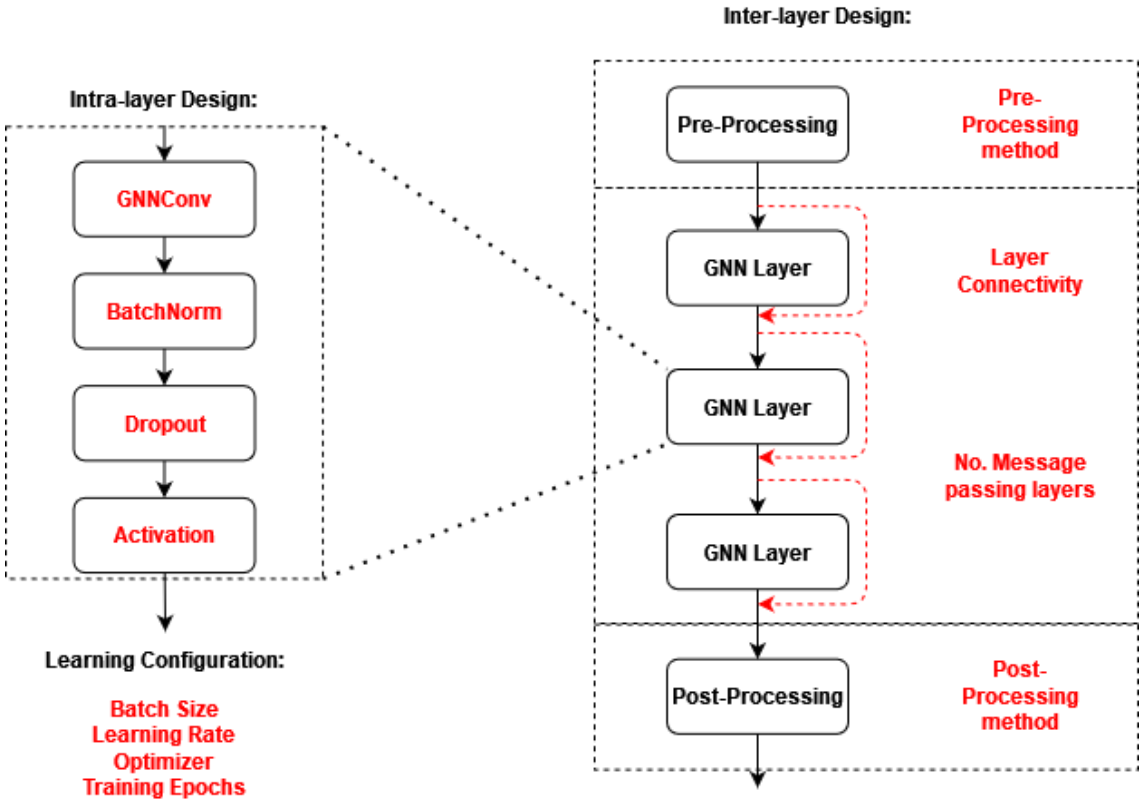


Figure 4: **Overview of the GNN design space.** A GNN design space consists of the intra-layer design, the inter-layer design and the learning configuration.

The focus of this work is on using an ID-GIN for modeling communication networks and estimating their performance metrics. The communication network is represented as a heterogeneous graph with different types of information attached to the nodes and edges. The ID-GIN attempts to learn a model of the graph-structured information conditioned on each node and edge type. The ID-GIN model is trained and evaluated with a dataset generated using the OMNet++ [22] network packet-level simulator, the final accuracy of the model is evaluated using the Mean Absolute Percentage Error (MAPE) metric computed on all the paths of the network:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}|\frac{\hat{y}_i - y_i}{y_i}|$$

where n is the number of samples and $\hat{y}$ is the network output and y is the true label. To verify that the model is able to generalize to unseen networks, the model is tested on topologies not seen during training. Since we also focus on the scalability of our solution the topologies used for testing are from considerably larger networks.

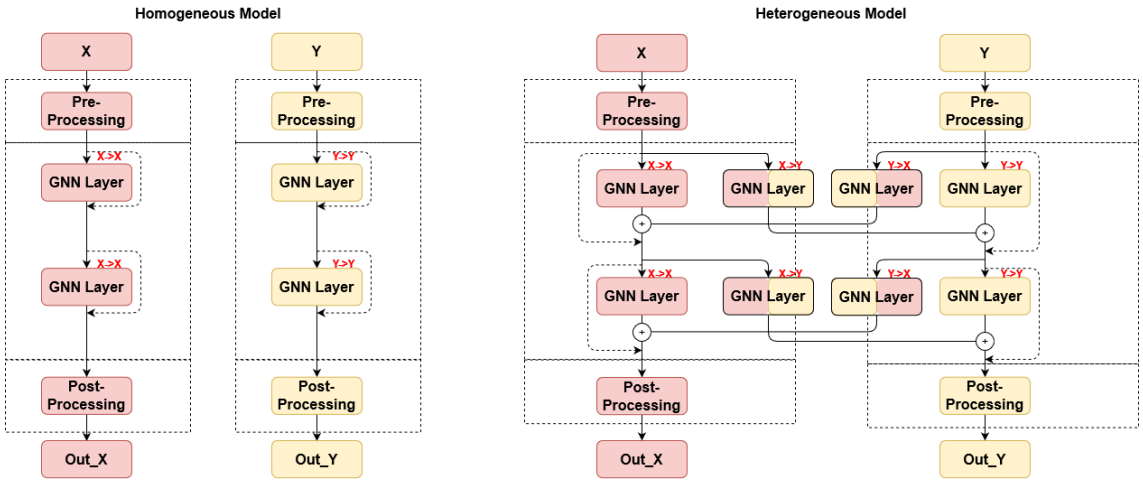


Figure 5: **Comparison Between Two Homogeneous GNN Models And Their Equivalent Heterogeneous GNN Model If Undirected Edges are Added.** To create a heterogeneous GNN model from a homogeneous GNN model, an additional GNN layer is added for each edge connecting different types of nodes.

Objectives

The key properties to have present in our proposed solution are accuracy, generalization and scalability.

- *Accuracy*: To accurately model a real world communication network we represent the communication network as a heterogeneous graph. We then use a ID-GIN conditioned on each node and edge type to learn and model the heterogeneous graph. This model is expected to take as input the network topology, routing configuration, and traffic, and output the average delay for each path in the network. The final accuracy of the model is determined using the mean absolute percentage error of all the paths of the network:

- *Generalization*: Training communication network models on real-world customer networks directly is usually not feasible, as it requires generating cases that may lead to service disruptions (e.g. link-failures). So the model should be able to generalize to networks not seen during training. Graph Neural Networks are the only machine learning based technique that has shown good generalization capabilities on different networks [23]. To test the generalization capabilities of the proposed solution we test the model using simulations generated from unseen network topologies and routing configurations.

- *Scalability*: Training communication network models on networks of similar size to real-world customer networks is computationally expensive. A possible solution is training on a smaller networks, and then deploying them in real-world networks. For that to be possible the model needs to be scalable. To test the scalability of the model we use a training set that contains simulations generated from network topologies containing 25-50 nodes and a test set that contains simulations from network topologies of size 51-300. However, this creates an issue because traditional machine learning methods are built on the assumption that the training and test sets are independent and identically distributed ($i.i.d$). Larger networks have some features with considerably different values than those of smaller networks (e.g. link bandwidth, path length), which creates a distributional shift between the training set

and test set. Producing out-of-distribution (OOD) values with neural networks is an open problem in the machine learning field. To circumvent this issue we try to only consider scale independent features such that both the training and test set follow a similar distribution.

Dataset

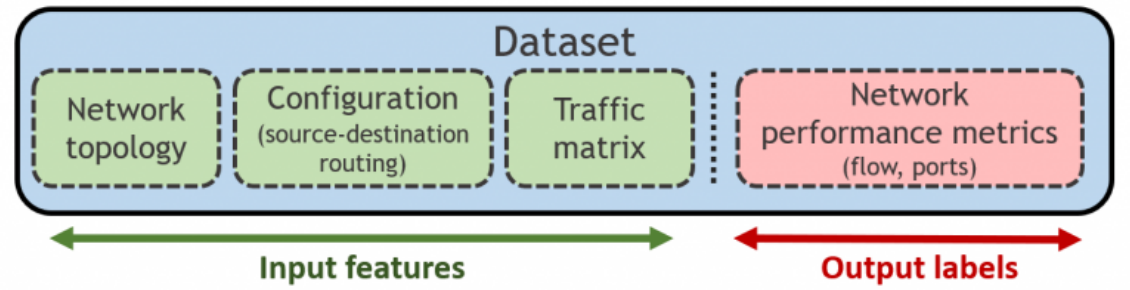
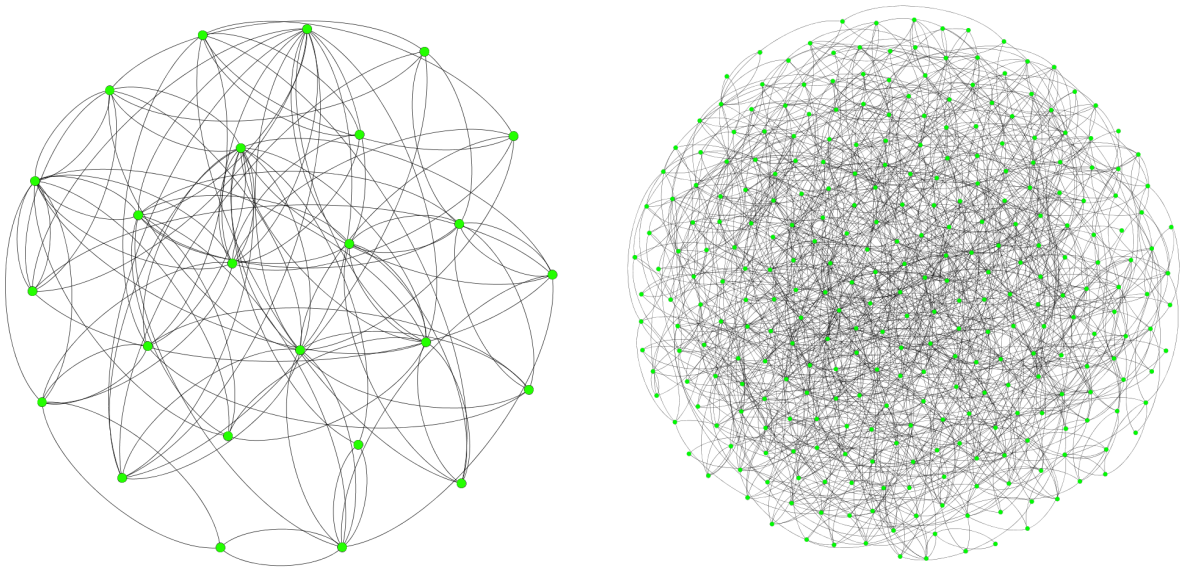


Figure 6: Schematic representation of the dataset [1]

The dataset used was generated using the OMNet++ [22] network simulator; it contains the network performance simulation results for different network configurations.The dataset consists of 124,680 simulations divided into training, validation, and test sets. The training set simulations are generated from network topologies of size 25-50 nodes. As the focus is on the scalability of the proposed solution the validation and test sets includes simulations from considerably larger network topologies of size 50-300. All the topologies have been artificially generated using the *Power-Law Out-Degree Algorithm* [24], where the ranges of the α and β parameters of the algorithm have been extrapolated from real topologies from the Internet Topology Zoo repository [25]. Fig.7 shows the topologies of two randomly selected samples from the training and test sets. A single network simulation is defined by the network topology, the routing configuration, the source-destination traffic matrix, and the network performance metrics, as shown in fig.6. The simulator implements a method that stops the simulation when it detects that the network has reached a stationary state. Consequently, the simulation time to generate each sample is different. This is why the performance measurements are provided as an average or as a number relative to time units. The Routing configurations were set using variations of the shortest path policy. In the validation and test set, some of the routing configurations were artificially set to longer paths, this was done to test the generalization capabilities of the network to longer paths.



(a) Sample from training set with 25 nodes

(b) Sample from test set with 300 nodes

Figure 7: Randomly selected inputs from the training and test sets

References Cited

[1] J. Suárez-Varela, M. Ferriol-Galmés, A. López, P. Almasan, G. Bernárdez, D. Pujol-Perich, K. Rusek, L. Bonniot, C. Neumann, F. Schnitzler, and et al., "The graph neural networking challenge," *ACM SIGCOMM Computer Communication Review*, vol. 51, no. 3, p. 9–16, Jul 2021. [Online]. Available: http://dx.doi.org/10.1145/3477482.3477485

[2] F. Ciucu and J. Schmitt, "Perspectives on network calculus: No free lunch, but still good value," vol. 42, no. 4, p. 311–322, aug 2012. [Online]. Available: https://doi.org/10.1145/2377677.2377747

[3] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," 2018.

[4] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," 2021.

[5] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," 2018.

[6] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and

Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1263–1272. [Online]. Available: https://proceedings.mlr.press/v70/gilmer17a.html

[7] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," *arXiv preprint arXiv:1905.11136*, 2019.

[8] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4602–4609.

[9] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," 2019.

[10] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2019.

[11] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI, Series*, vol. 2, no. 9, pp. 12–16, 1968.

[12] L. Babai and L. Kucera, "Canonical labelling of graphs in linear average time," in *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. IEEE, 1979, pp. 39–46.

[13] J.-Y. Cai, M. Fürer, and N. Immerman, "An optimal lower bound on the number of variables for graph identification," *Combinatorica*, vol. 12, no. 4, pp. 389–410, 1992.

[14] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 1365–1374.

[15] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[16] J. You, J. Gomes-Selman, R. Ying, and J. Leskovec, "Identity-aware graph neural networks," 2021.

[17] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456. [Online]. Available: https://proceedings.mlr.press/v37/ioffe15.html

[18] P. Baldi and P. J. Sadowski, "Understanding dropout," *Advances in neural information processing systems*, vol. 26, pp. 2814–2822, 2013.

[19] P. Sibi, S. A. Jones, and P. Siddarth, "Analysis of different activation functions using back propagation neural networks," *Journal of theoretical and applied information technology*, vol. 47, no. 3, pp. 1264–1268, 2013.

[20] K. Xu, C. Li, Y. Tian, T. Sonobe, K. ichi Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," 2018.

[21] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 793–803. [Online]. Available: https://doi.org/10.1145/3292500.3330961

[22] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008, pp. 1–10.

[23] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of graph neural networks for network modeling and optimization in sdn," *Proceedings of the 2019 ACM Symposium on SDN Research*, Apr 2019. [Online]. Available: http://dx.doi.org/10.1145/3314148.3314357

[24] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *The Structure and Dynamics of Networks*. Princeton University Press, 2011, pp. 195–206.

[25] "The Internet Topology Zoo," http://www.topology-zoo.org, accessed: 28-11-2021.