A Deep Learning Approach to Topology Configuration in Multi-Hop Wireless Networks with Directional Antennas: nodes2net

Félix Marcoccia, Cédric Adjih, Paul Mühlethaler

A Deep Learning Approach to Topology Configuration in Multi-Hop Wireless Networks with Directional Antennas: nodes2net

Félix Marcoccia[1], Cédric Adjih[2], and Paul Mühlethaler[3]

[1]INRIA Paris, Sorbonne Université, Thalès SIX GTS, Paris, France
[2]INRIA Saclay, Palaiseau, France
[3]INRIA Paris, Paris, France

***Abstract*— Multi-hop wireless networks can be optimized using directional antennas, as they allow for in-depth interference management and network topology optimization. This type of optimization involves ensuring high operational guarantees such as instantaneous connectivity, minimum SNRs and SINRs thresholds, and improved QoS. It simplifies tasks of future network layers and allows for more relaxed routing protocols and scheduling. However, attaining optimal performance via network configuration involves selecting an antenna orientation for each node to create a link with another node. This is challenging, especially when the process is carried out in real-time. To tackle this challenge, we present nodes2net, a Deep Neural Network (DNN) that is trained to imitate solved, ideal network instances. This approach uses nodes' positions as inputs and produces a set of links as output. By leveraging learning of patterns and theoretically driven properties, nodes2net can generate reliable network configuration solutions when dealing with new sets of node positions. It utilizes efficient neural network aggregation operators to facilitate and process information about the nodes, to finally produce the final solution as set of links. Our results demonstrate the competitive performance of this method.**

I. Introduction

Multi-hop wireless networks, especially with the current deployment of 5G technologies and research work on 6G, constitute a very active field of research. While the per-user capacity and throughput of a network are known to scale poorly with the increase of users [1], it has been proven that using directional antennas with reduced beamwidth can help mitigate the loss in Quality of Service (QoS) by a factor inversely proportional to the beamwidth [2].

Using directional antennas allows for both higher per-link throughput and much better interference management. One must chose wisely, for each antenna, towards which node to point. In effect, we want to steer all the antennas in a way that avoids creating low SNR links, and high interference patterns. In order for the antennas to be aligned at transmission time, paths from users to others have to be computed in advance, either in the form of routes computed along with the resource allocation and traffic control level (OSI layer 3), or even beforehand to reduce later computations, by computing carefully an optimized network topology (OSI layer 2) to allow for easier network and traffic management afterwards. Usually, on top of the network constituted by the links, proper routing and scheduling need to be computed. An example is given by [3], that assumes a slotted frame structure, as well as some known antenna orientations, and that, on that basis, computes a routing table, and then establishes a transmission schedule. The most essential constraint is connectedness; moreover, the network must respect several physical constraints. We observe that this problem has a highly combinatorial aspect: global connectedness relies on complex combinations of links, and physical constraints require taking into account interdependence and interference between created links. We aim to create a topology that provides instantaneous global connectedness and fair adherence to physical constraints to alleviate further network tasks. To mitigate some of the combinatorial and computational burdens, we propose a one-step process that generates such a topology. In this article, we address the most important issue, connectedness, and focus on configuring the network, e.g., steering antennas, which we entirely solve using deep learning techniques. For this purpose, we train a Deep Neural Network to imitate the results of an Integer Programming (IP) instance of the problem. It can then be used to infer graphs that hold the same properties as the ones labeled as solutions of the IP problem, in constant time The rest of the article is organized as follows: in Sec. II, we present articles and literature related to optimizations of such networks. In Sec. III, we detail the system model and our problem statement. Then we introduce our solution and its properties in Sec. IV-A. In Sec. V we present some results that confirm the value of our method and illustrate how its components impact its performance, thus proving their merit. We finally conclude in Sec. VI.

II. Related Work

The computation of efficient network solutions has recently been the target of great effort, especially in the context of the rise of 5G and 6G technologies. While mathematical formalization via Integer Programming is often used to obtain optimal solutions, with typical path, links and flows linear formulations such as in [4], one must also often derive heuristics to enable faster solving such as in [3]. While this

notably accelerates the computation of solutions, the process is still most of the time iterative, often based on greedy algorithms and often relies on an iterative process, which can be problematic in some operational applications. A key advantage of Deep Learning methods is that the generation of a solution involves a straightforward series of matrix multiplications, with coefficients that have been pre-learned, simplifying the overall computational process. Using Deep Learning algorithms for such tasks can be seen as a way to turn combinatorial problems into non linear and multivariate parametrized statistical problems. Whether we deal with link or graph-level prediction, clustering or network performance prediction, such difficult problems require a fine use and complex combinations of the inputs. Another positive aspect of Deep Neural Networks is that they scale well with the number of users, as their complexity can be made to scale linearly with the number of inputs nodes, where combinatorial methods can struggle to handle high number of inputs nodes.

While they can be used on simpler tasks such as a network performance prediction, e.g. in [5], Deep Neural Networks can also be used to infer network graphs, often in a dynamic, temporal prediction manner as in [6]. Networks are graphs, and Deep Learning for graphs has been receiving strong attention lately, as Graph Neural Networks (GNN) [7, 8] have become one of the hottest topics of research in the Deep Learning community. Generative models for graphs such as GraphVAE [9], which embeds the whole graph thanks to a GNN and a global pooling operation, have been presented and allow for continuous and possibly conditioned graph generation. When dealing with graphs, we need nodes to be able to gather information about other nodes and to organize this information to get a better knowledge of the situation of the nodes. In neural networks, this is usually the role of the *feature aggregator*, which is the operator that we train to combine signal coming from the data objects in a relevant way. It enables the learning of high level features that help solving the problem. What is problematic is that nodes come with a totally arbitrary node ordering, that must not be taken into account by a feature aggregator that would treat the set of nodes as an ordered sequence, processing each node depending on their position in the sequence and combining their features accordingly. The whole challenge of learning on graphs is hence finding a learnable, permutation invariant, feature aggregator that allows nodes to communicate information, which GNNs do with edge-conditioned convolutions or any edge-conditioned message passing operators. In our problem, since we do not have edges, as they are the objects we wish to infer, we need another feature extractor that follows the same type of guidelines as GNNs. The famous Attention mechanism is a perfect candidate for that. It has mainly been introduced in [10], while [11] popularized it even further with a retrieval-system-like formulation and empirically proved its expressivity with the Transformer model.

III. System Model

We consider a set V of n nodes described by their respective 2D coordinates (x, y). We assume an idealized "protocol model", where nodes can transmit iff they are within the radio range. Each node has a fixed number of independent antennas, each of which can establish at most one link. Candidate links correspond to pairs of nodes that are in range of each other, and hence are represented as undirected edges. Our problem is to find a subset of those edges, denoted E, that satisfies some constraints while possibly optimizing some objective function f. E is exactly the links obtained after orienting (configuring) antennas. Once the edges are given, we then have a graph $G = (V, E)$ that can serve as the network link topology. As a comprehensive example, and throughout this work, we will consider the problem of the creation of a network of n nodes under some physical link constraints (limited number of communication links per user, a fixed maximum length for each link...) that must ensure global connectedness while minimizing the total number of links. It can be viewed as an optimization problem with several constraints corresponding to physical limitations of the communication links. We formalize our problem as an Integer Programming one to obtain optimal solutions, and will train a neural network to output graphs as close as possible to these solutions.

Our problem can be formalized as follows :

- V is the set of nodes 1,2,...,n
- $e_{i,j}$ is a binary decision variable that indicates that there is an edge between node i and node j (i.e. after orientation of one of their antennas to create this link)
- One node is also described by its 2D coordinates x_i, y_i

We want to solve the following optimization problem

$$\min_{e_{i,j}} \sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} \tag{1}$$

s.t.

Logical and physical constraints (2)-(9) :

One node holds n_{antennas}. Since one antenna can not form several links one node can then form at most N_{antennas} links :

$$\forall i \in V, \quad \sum_{j=1}^{n} e_{i,j} \leq N_{\text{antennas}} \tag{2}$$

One link can be formed between two nodes only if they are within a maximum radio range D_{max} one from the other:

$$\forall i, j \in V^2, \quad e_{ij} \times [(x_j - x_i)^2 + (y_j - y_i)^2] \leq D_{max} \tag{3}$$

Links are bidirectional:

$$\forall i, j \in V^2, \quad e_{i,j} = e_{j,i} \tag{4}$$

One node can not form a link with itself:

$$\forall i, j \in V^2, \quad i = j \Rightarrow e_{i,j} = 0 \tag{5}$$

In addition, it is necessary to establish a mathematical formalization for the connectivity of the entire network. A commonly used technique is to introduce phantom flows originating from a virtual source at a specific node (without loss of generality, let's denote it as v_0). These phantom flows are required to traverse through every node in the network, and flow conservation equations are formulated accordingly. The network is considered connected if such flows can be identified. The constraints that enforce connectedness through phantom flows are as follows:

Virtual source distributes $n-1$ flows through the network:

$$\sum_{j=0}^{n} f_{0,j} = n - 1 \tag{6}$$

Flows propagate through existing links:

$$\forall i, j \in V^2, \quad i \neq j \Rightarrow f_{i,j} \leq (n-1) \times e_{i,j} \tag{7}$$

Each node absorbs one flow and transmits the rest:

$$\forall i \in V, \quad \sum_{j=0}^{n} f_{i,j} = \sum_{j=0}^{n} f_{j,i} - 1 \tag{8}$$

One node can not send a flow to itself:

$$\forall i, j \in V^2, \quad i = j \Rightarrow f_{i,j} = 0 \tag{9}$$

IV. Our Approach: nodes2net

A. Model

In this paper, we propose *nodes2net*, an Attention-based Deep Neural Network that produces a set of network links from the nodes' positions.

Our model takes as inputs the set V of nodes described as:

$$v_i = (x_i, y_i) \quad \forall i \in [1, n].$$

It outputs an adjacency matrix:

$$A_{pred} = \begin{pmatrix} e_{11} & e_{12} & \cdots & e_{1n} \\ e_{21} & e_{22} & \cdots & e_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{n1} & e_{n2} & \cdots & e_{nn} \end{pmatrix}$$

that describes the set of edges E of the desired graph.

We want to find a function F, namely a Deep Neural Network, parametrized by its weights θ, such that

$$F_\theta(V) = A_{pred}.$$

In the training phase, we first solve the optimization problem (1)-(9) on some random instances of graphs through a MILP solver (after linearization of the equations). We use these solved instances to constitute the dataset mapping input nodes' positions to their respective desired topology of links. For each datapoint of nodes' positions V this hence yields the label adjacency matrix corresponding to the true solution called A_{pred}, that the model should then "predict". We then want to find the best model to learn this mapping between nodes and topologies.

Challenges: While a Multi-Layer Perceptron (MLP) may initially seem like a straightforward network architecture well suited for such problems, it is biased by the ordering of the nodes. One main issue with creating a graph from nodes is indeed the need for a good permutation-invariant feature aggregator between nodes. While GNNs use edge-conditioned, spectrally defined convolutions, or message passing algorithms, they are not suited for a whole graph prediction task since the edges are yet to be predicted, hence not usable to serve as signal propagation intermediates. Another mentionable aspect is that, since we are dealing with rather small spatial graphs, using pooling and global aggregators to capture structural and high level features might not be entirely suitable, since we wish to find precise features at node level that capture local dependencies and interactions between nodes given their position. We must also keep node-level features instead of whole graph-level features in most of the network since we must condition the signal on nodes' position, and allow the nodes to exchange information accordingly.

Our Architecture: We use the Attention mechanism [11] as the main permutation invariant feature aggregator. We also use residual connections [12] from the input to further layers not to lose the position signal of each node (in practice we rather concatenate the positions to further layers instead of adding them, as it showed to be more efficient, they are hence not textbook residual connections) and between layers not to lose each node's individual embedding. The simplified embeddings of the set of nodes V going through one Attention layer l (for one Attention head, in practice we use up to 8) can be described as below:

$$V_{l+1} = V_l + \text{softmax}\left(\frac{W_q Q W_k K^T}{\sqrt{d_k}}\right) W_v V_l \oplus V_0$$

where V_0 represents the positions of the inputs nodes, V_l the embeddings of the nodes at layer l and $\oplus$ is either a dimensionally arranging addition (because the inputs and the embeddings do not have the same dimension) or a concatenation. Q is the pack of Queries formulated by the nodes, K the pack of the Keys that the nodes use to describe themselves and V the pack of Values corresponding to the embeddings of the nodes. W_q, W_k and W_v are learnable matrices that are what we train to get the Attention to capture the relevant features. We use MultiHead Attention, which defines several different instances of these learnable weights to allow attending to different characteristics, then aggregates them.

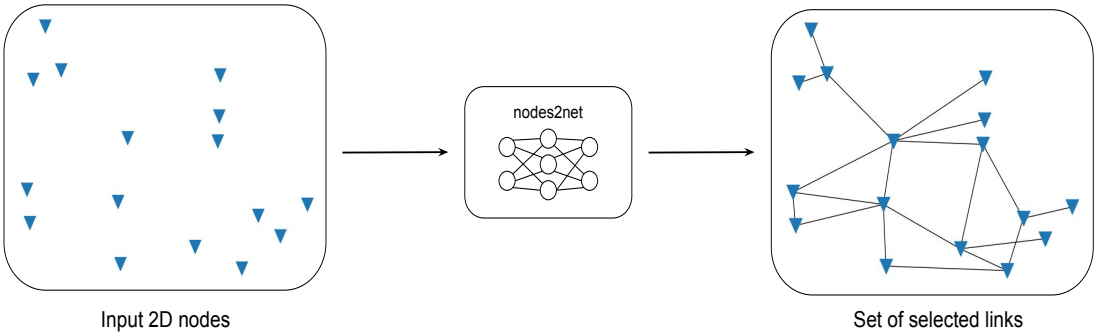

Figure 1: *Overview of our solution: the model takes the nodes' positions as input and outputs the set of edges to connect them. The 2D positions of the nodes and the selected links correspond on one actual dataset sample and its proposed solution.*

The idea is that one node v_j which is relevant to attend to for a node v_i will learn to define a $W_k K_j$ rather similar (to maximize $W_q Q_i W_k K_j$) to $W_q Q_i$ such that v_j "wins" the softmax i.e. v_i "understands" that it must attend to node v_j. The point is then to learn a parametrization of the weight matrices W_q, W_k in order to learn a reliable transformation that captures relevant pairs of nodes so that they present well-aligned Query and Key. The Value weight matrix W_v learns how to finally embed the node combining the attended nodes information.

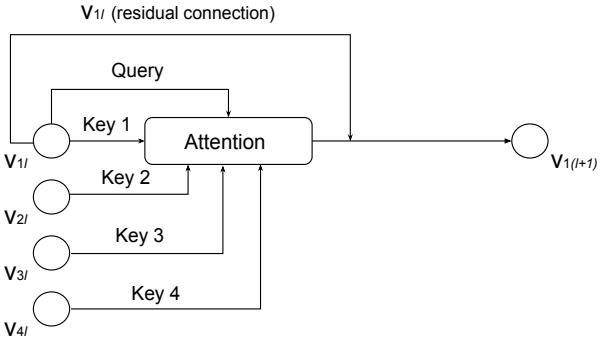

Figure 2: Illustration of one embedding layer applied to one node v_1 in a 4-node-network

The network is composed of a succession of such Attention layers connected by Linear layers, that we notably use to increase the dimension of the signal. We use the first Linear layer to increase the dimension of the signal fed to the Attention layer. In our experiments, we used two Attention layers since it offered good performance and kept the model rather light. We top the network with a flattened Fully-Connected layer as the prediction layer, which combines the learned nodes embeddings to make the adjacency matrix prediction.
While the Attention-based feature extractor is permutation invariant, the prediction layer, similar to the prediction layers of typical ConvNets [13], is not. We found that it still offers good performance and generalization, and induces a practical node ordering bias that allows to organize easily edges as an adjacency matrix.
We also implement a permutation invariant prediction layer that consists in computing Attention scores and in feeding them into a few Linear layers to get the per-node link predictions, but the method does not do as well as the previous one, especially in terms of generalization (as showcased in Table I, and II). Furthermore, more costly Graph Matching algorithms that still allow the gradients to flow can be used for similar tasks [9] but demand higher effort to align predicted nodes and edges with labeled ones.

B. Graph-level attributes

We can also add graph-level attributes by adding them as inputs and concatenating them with the nodes, allowing the nodes to attend to them, using FiLM Layers [14][15] or a HyperNetwork [16] architecture. We then search F parametrized by θ such that

$$F_\theta(V, \gamma) = A_{pred}$$

where γ is a sequence of graph level attributes. Graph attributes can be for example several spectral properties such as the algebraic connectivity, or can be related to the density of links, centrality, and can also be learned. The respect of the desired graph attribute in the predicted matrix can be enforced using Lagragian Duality framework described in [17] as long as they are differentiable w.r.t. the weights of the model.

C. Deterministic formulation: Variational Inference

The problem itself is not deterministic, which might lead to unstable training in the supervised learning setting, since one correct output of the model can cause great error. It has not been extremely problematic in practice, since, while the problem is not deterministic, the dataset is, and the model training still seems to converge rather easily. We still derive a Variational Inference method, similar to a Variational Autoencoder version of the network, so that the problem becomes a reconstruction, hence deterministic, problem. The learning problem consists in feeding the graph to the encoder, compressing it into a continuous and low dimensional latent space, then decoding the compressed signal to reconstruct the inputs. We condition the decoder on the positions of the nodes. Infering a new graph then consists in sampling in a continuous latent space, adding the nodes' positions as

a conditioning signal and feeding them to the decoder. To enable sampling, and thus generation, we want a smooth, continuous latent space. To do so, we enforce the encoder's output to follow a multivariate gaussian distribution. On the other hand, we train the decoder to output values close to the input being reconstructed. The encoder consists in a graph feature aggregator, which can be a GNN, or a "classical" model based on Laplacian Eigenmaps PE [18] for instance. We can either concatenate or attend to this encoded vector with the positions of the nodes as a conditioning signal, then use Attention in the same way we used it in the supervised version, allowing for permutation invariant feature agregation of the nodes encodings, and globally follow the same architecture as described above.
The problem then results in maximizing the following:

$$\mathcal{L}(\theta, \phi; \mathbf{G}^{(i)}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{G}^{(i)})}\left[\log p_\theta(\mathbf{E}^{(i)}|\mathbf{V}^{(i)}, \mathbf{z})\right] - D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{G}^{(i)})||p(\mathbf{z})\right)$$

q_ϕ represents the approximated posterior latent distribution
p_θ represents the approximated likelihood of the data
$p(z)$ represents the prior distribution of the latents, we assume it to follow a multivariate Gaussian distribution.
The first term is the Reconstruction Loss, maximizing it enforces the reconstructed edges to be as close as possible to the input edges. The second term is the Kullback-Leibler Divergence, minimizing it enforces the approximated latent distribution to be close to the prior distribution z that is assumed to follow a multivariate Gaussian.
Infering a graph prediction hence consists in the network estimating:

$$decoder(\mathbf{V^{(i)}}) = p_\theta(\mathbf{E^{(i)}}|\mathbf{V^{(i)}}, \mathbf{z}).$$

The idea that motivates this formulation is that, despite the strong signal brought by the position of the nodes, the variational encoder can still provide sufficient information to distinguish one particular solution. It is useful considering an asymptotically large dataset, where each set of positions admits many labeled solutions. In our case, we observed that the model mainly learns a mapping from nodes' positions to predicted links without making great usage of the encoded vector.

V. Experimental Results

We conduct our experiments with an Intel Xeon(R) E5-2650v3 at 2.30 GHz CPU and a Tesla T4 GPU.
With such hardware, inference time does not exceed 1 ms even with the largest versions of the model. Our method is implemented using PyTorch. We use AdamW optimizer with weight decay rates between 0.1 and 0.15, a learning rate of 1e-4, a batch size of 64, for 100 epochs. We use a dataset of multiple instances with randomly generated nodes' positions. It is composed of 100k samples of 16 nodes' positions and the adjacency matrix obtained solving the IP problem with a solver (namely Gurobi). We also conduct another experiment on 8 nodes graphs where the network is built iteratively starting from a random node and respecting some more strict geometrical constraints that simulate the positions of the antennas on the nodes. The dataset is more diverse as, for instance, some graphs are not entirely connected. This dataset is composed of 800k graphs.

Table I: Results on dataset with 100k samples of 16 nodes. We can see that nodes2net outperforms a simple baseline as well as the positive effect of each component on performance.

Model	val acc	test acc
nodes2net	95.61 %	89.77 %
nodes2net (no residual connections)	94.87 %	88.81 %
nodes2net (attention score prediction)	92.29 %	84.83 %
MLP (flattened)	90.31 %	84.73 %

Table II: Results on the iteratively-constructed 8 nodes dataset. We observe slightly better performance than on the IP dataset and similar observations as regards the comparison with the baseline and the impact of the components.

Model	val acc	test acc
nodes2net	96.08%	90.82 %
nodes2net (no residual connections)	95.14 %	89.56 %
nodes2net (attention score prediction)	92.71 %	85.21%
MLP (flattened)	90.59 %	85.11 %

The accuracy is measured as in a classical link prediction task for each link: we round the output of the network to either 0 or 1 and we measure the difference with the true label for each entry of the adjacency matrix. The dataset of Table I contains almost 80% zeros, it is easier to predict the absence of link than to predict a link, so this has to be taken into account when reading the accuracy scores. The 8 nodes iterative dataset of Table II contains around 65% zeros, in theory, it should then be harder to predict in that regard, but it does not show there. We can hypothesize that the stricter geometric constraints are well captured and help reduce the number of possible links for each node, hence making the prediction task easier. Test set graphs are not always reproduced well but it is not necessarily a problem since, as we mentioned before, several graphs can be solutions for a single set of nodes, and the applicative goal of the paper is to generate plausible graphs with similar properties as the ones learned. We can still sometimes observe, almost exclusively when dealing with never seen nodes' positions, and more often with 16 nodes scenarios than with 8, some impossible, illogical links that violate some constraints or seem really sub-obtimal given the optimization task. Larger prediction layer can help prevent such issues but imply greater computation cost and careful training to avoid overfitting. Error management in such DNNs can be challenging because of the "one-shot" nature of the inference (all links are created in a single computation stage) but can still be addressed using hardcoded security checks. While the permutation invariance guarantees

better generalization properties of the learned features, we also conducted multiple experiments to evaluate the impact of using the Attention mechanism instead of a simple MLP with several flattened layers on the training itself.

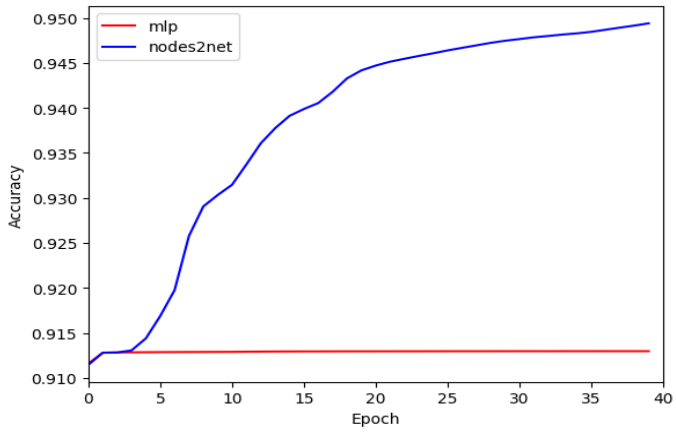

Figure 3: Convergences of nodes2net (attention-based) and MLP visualized after the sigmoid convergence. We can see that nodes2net continues to converge durably after the standard flattened MLP stagnates.

In Fig. 3 we visualize the per-epoch averaged accuracy of the model after the "sigmoid drop", the MLP struggles to gain more accuracy while the attention model shows much better figures and a rather stable convergence, we conducted this experiment with and without residual connections and the tendency was similar in both cases (note that the values are slightly different than those from Table I because we only train them for 40 epochs and the accuracy is computed live without rounding the prediction values to 0 or 1). We can deduce that the Attention mechanism captures characteristics that could not be captured with the MLP and hence allows for much more accurate predictions.

VI. Conclusion

With nodes2net, we provide an effective framework to allow for a constant time and flexible network creation, which can be really useful, especially in the here described field of wireless network topology optimization, where solving optimization problems on the go, or using online convergence algorithms, are not possible. Using Attention between nodes as a feature extractor suits our problem well and allows to efficiently capture features between nodes while respecting permutation invariance and without suffering from oversmoothing of the input signal. We derived a variational inference formulation that overcomes the theoretical limit of supervised learning for a non-deterministic problem. The good inductive bias that our model seems to present are confirmed by our experiments. Future work will aim at improving even further the generalization capabilities and allowing to enforce strict respect of constraints. Future work will also consist in more applicative studies to jump from our Deep Learning method to a whole operational network solution and measure the performance and the QoS we can expect from it. We will also work on the integration of such a neural network on embedded devices. Given that the model is small and can be parallelized, and in view of the recent advances in network quantization and pruning, real time graph prediction seems realistic.

References

[1] P. Gupta and P. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, Mar. 2000. [Online]. Available: http://ieeexplore.ieee.org/document/825799/

[2] S. Yi, Y. Pei, and S. Kalyanaraman, "On the capacity improvement of ad hoc wireless networks using directional antennas," in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, ser. MobiHoc '03. New York, NY, USA: Association for Computing Machinery, Jun. 2003, pp. 108–116. [Online]. Available: https://doi.org/10.1145/778415.778429

[3] A. Benhamiche, W. da Silva Coelho, and N. Perrot, "Routing and Resource Assignment Problems in Future 5G Radio Access Networks," Jun. 2019.

[4] W. Feng, Y. Li, D. Jin, L. Su, and S. Chen, "Millimetre-Wave Backhaul for 5G Networks: Challenges and Solutions," *Sensors*, vol. 16, no. 6, p. 892, Jun. 2016, number: 6 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/1424-8220/16/6/892

[5] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2260–2270, Oct. 2020, conference Name: IEEE Journal on Selected Areas in Communications.

[6] K. Lei, M. Qin, B. Bai, G. Zhang, and M. Yang, "GCN-GAN: A Non-linear Temporal Link Prediction Model for Weighted Dynamic Networks," Jan. 2019, arXiv:1901.09165 [cs]. [Online]. Available: http://arxiv.org/abs/1901.09165

[7] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan. 2009, conference Name: IEEE Transactions on Neural Networks.

[8] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, Jul. 2017, arXiv:1611.08097 [cs]. [Online]. Available: http://arxiv.org/abs/1611.08097

[9] M. Simonovsky and N. Komodakis, "GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders," Feb. 2018, arXiv:1802.03480 [cs]. [Online]. Available: http://arxiv.org/abs/1802.03480

[10] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," May 2016, arXiv:1409.0473 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1409.0473

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," Dec. 2017, arXiv:1706.03762 [cs]. [Online]. Available: http://arxiv.org/abs/1706.03762

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 2015, arXiv:1512.03385 [cs]. [Online]. Available: http://arxiv.org/abs/1512.03385

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, conference Name: Proceedings of the IEEE.

[14] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," 2017.

[15] M. Brockschmidt, "Gnn-film: Graph neural networks with feature-wise linear modulation," 2020.

[16] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," 2016.

[17] F. Fioretto, P. V. Hentenryck, T. W. Mak, C. Tran, F. Baldo, and M. Lombardi, "Lagrangian duality for constrained deep learning," 2020.

[18] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," 2022.