Attentive, Permutation Invariant, One-Shot Node-conditioned Graph Generation for Wireless Networks Topology Optimization

Félix Marcoccia, Cédric Adjih, Paul Mühlethaler

Attentive, Permutation Invariant, One-Shot Node-conditioned Graph Generation for Wireless Networks Topology Optimization

Félix Marcoccia[1,2,4], Cédric Adjih[3], and Paul Mühlethaler[1]

[1] INRIA Paris, Paris
[2] Thales SIX, Gennevilliers, France
[3] INRIA Saclay, Palaiseau, France
[4] Sorbonne Université, Paris, France

Abstract. It is common knowledge that using directional antennas is often mandatory for Multi-hop ad-hoc wireless networks to provide satisfying quality of service, especially when dealing with an important number of communication nodes [1]. As opposed to their omnidirectional counterpart, directional antennas allow for much more manageable interference patterns: a receiving antenna is not necessarily interfered by nearby emitting antennas as long as this receiving antenna is not directed towards these undesired emission beams. Two nodes then need to steer one of their antennas in the direction of the other node in order to create a network communication link. These two users will then be able to, in turn, emit and receive to and from each other. The scope of this work resides in finding a centralized algorithm to governate these antenna steering decisions for all users to instantaneously provide a valid set of communication links at any time given the positions of each user. The problem that raises is then a geometrical one that implies finding topologies of network links that present satisfying throughput and overall QoS and guarantee instantaneous connectedness i.e. the computed set of links allows any user to reach any other user in a certain number of hops. Building such optimized link topologies makes further tasks, such as routing and scheduling of the network, much simpler and faster. This problem is highly combinatorial and, while it is solvable with traditional Mixed Integer Programming (MIP), it is quite challenging to carry it out in real time. For this purpose, we propose a Deep Neural Network that is trained to imitate valid, solved instances of the problem. We use the Attention mechanism [2] [3] to let nodes exchange information in order to capture interesting patterns and properties that then enable the neural network to generate valid network link topologies, even dealing with unseen sets of users positions.

1 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 [4], 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 [1]. 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. Indeed, 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 [5], 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. Some important network properties have to be guaranteed, the most crucial one being 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. In an ideal scenario, where the topology is optimal, routing can then be simple shortest paths and scheduling can consist in simple 2-slot scheduling where one half of the nodes emits while the other half receives and vice versa. This is of course a complex problem to tackle. To mitigate some of the combinatorial and computational burdens, we propose a one-step process that generates such a topology. 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.

Graphs are heterogeneous objects, composed of both nodes and edges, which do not have any natural ordering. They then require an Neural Network architecture that can feature both node and edge computations and respect some crucial properties not to bring undesirable biases due to the nodes' or the edges' ordering. This is the main focus of our work. The rest of the article is organized as follows: in Sec. 2, we present articles and literature related to optimizations of such networks. In Sec. 3, we detail the system model and our problem statement. Then we introduce our solution and its properties in Sec. 4.1. In Sec. 6, 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. 7.

2 Related Work

Recently, there has been a significant push to develop more effective network solutions, particularly with the emergence of 5G and 6G technologies. Integer Programming is commonly employed for finding optimal solutions, often utilizing linear formulations for paths, links, and flows as seen in sources like [6]. However, to quicken this process, different algorithms and heuristics are frequently implemented, as in [7] [5]. Despite speeding up solution computation, these methods are usually iterative and rely heavily on greedy algorithms, which can be a drawback in certain practical applications.

Deep Learning offers a notable advantage in this context. It simplifies the solution generation process to a sequence of matrix multiplications using pre-trained coefficients. This approach effectively transforms combinatorial challenges into non-linear, multivariate statistical problems with parametric nuances. Deep Learning is particularly beneficial because it scales efficiently with user numbers, maintaining linear complexity relative to the input nodes - a stark contrast to the scalability issues faced by combinatorial methods with numerous input nodes.

Deep Neural Networks (DNNs) are versatile and can be applied to both simple tasks like network performance prediction (as in [8]) and more complex ones like inferring network graphs, often through dynamic temporal prediction methods (as in [9]). The field of Deep Learning for graph structures, including Graph Neural Networks (GNN) [10][11], has been gaining immense attention. Generative graph models like GraphVAE [12], which use a GNN and a global pooling operation to represent entire graphs, facilitate continuous and potentially conditioned graph generation. GNNs typically offer edge-conditioned convolution or message passing techniques. In our case, without actual edges to analyze, we employ an alternative feature extractor aligned with some of the main GNN principles that ensure permutation invariance. Here, the well-known Attention mechanism, first introduced in [2] and popularized through its use in the Transformer model as demonstrated in [3], becomes highly relevant. This mechanism offers a promising approach, akin to a retrieval system, and its effectiveness has been empirically validated. Attention is generally popular to treat graph-shaped data, as [13] [14]. Attention could also benefit from some of the expressive power proved in [15][16] due to its Query/Key Retrieval formulation that implies some form of successive node-matrix multiplications in the layers. As a solution for node-conditioned network topology generation, nodes2net [17] was proposed, the authors use Attention and a flattened linear layer to get link predictions. While it gives good results, the generalization lacks theoretical guarantees and can be inconsistent because of the flat linear layer, which can be greatly biased by the ordering of the nodes. The fixed-sized linear layer also implies that the model is not flexible regarding the number of communication nodes.

3 System Model

The system we wish to study is the same as the one in [17]. 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 if 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 (antenna number, placement, more importantly connectedness...) while possibly optimizing some objective function f. E amounts to 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 connectedness 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:

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

4 Our Approach

We previously mentioned that carrying out such Integer Programming problems in real time can be difficult, especially with an important number of communication nodes. This is why our goal is to be able to extract the essence of the distribution of the solution graphs with a monolithic structure that can then generate new valid graphs when dealing with new, unseen sets of nodes.

4.1 Model

In this paper, we propose using Deep Neural Networks to infer valid sets of links, using the nodes' positions as inputs, by "imitating" optimal instances obtained

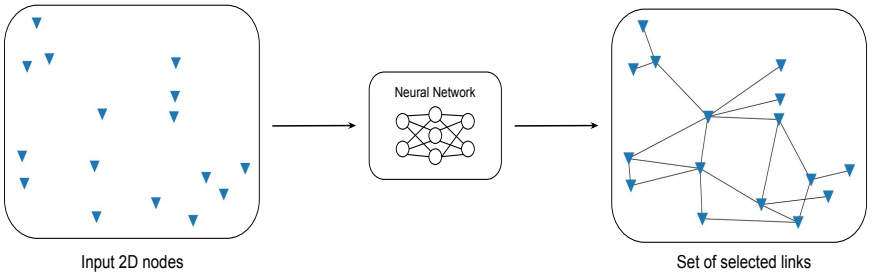

Fig. 1: *Overview of our solution: the model takes the nodes' positions as input and outputs the set of edges to connect them. The 2D positions of the nodes and the selected links correspond on one actual dataset sample and its proposed solution.*

by solving the aforementioned linear program.
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:

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

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

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

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

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 E_{pred}, that the model should then "predict". We then want to find the best model to learn this mapping between nodes and topologies.

Permutation Invariance and Attention
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 feature aggregator that does not process nodes as an ordered sequence,

which would for instance result in a neuron learning some combination of input 1 with input 2 and 4, which would not make sense and would obviously lead to undesirable bias since, as stated before, nodes do not feature any natural ordering. In general, achieving permutation invariance, which involves processing the graph at node-level, also allows a single neural network to be able to work with any given number of input nodes. This particularly useful for embedded environments where storage is limited, and to make sure that topologies are infered following the same computations regardless of the number of communication nodes. The ability for a feature aggregator to process information and capture patterns without being biased by the objects ordering is called permutation invariance (at the object level) of permutation equivariance (at the whole graph prediction level), both terms refer to a similar concept in the case of deep learning on graphs. While GNNs use edge-conditioned, spectrally defined convolutions, or strict message passing algorithms, most of them are not perfectly suited for a whole graph prediction task since the edges are yet to be predicted, and hence do not hold information that can help the prediction process. The global guideline a permutation invariant aggregator has to follow is to process at node and possibly edge-level, in order to capture some underlying local distribution that explains the link patterns of the dataset's graphs. Through the layers, the feature aggregation process should be able to capture larger patterns, involving more nodes and broader neighbourhoods of nodes, making the patterns somewhat "global" while keeping the computation at node and edge-level.
We use the Attention mechanism [2] [3] as the message-passing modality of our model. It enables nodes to exchange information in a way that allows fine feature aggregation amongst several nodes without breaking permutation invariance. We also use residual connections [18] from the input to further layers not to lose the position signal of each node and between layers not to lose each node's individual embedding. The most popular version of Attention is the one used for Transformer models [3], 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:

$$H_{l+1} = \text{softmax}\left(\frac{W_q H_l (W_k H_l)^T}{\sqrt{d_k}}\right) W_v H_l \oplus H_l \oplus H_{l-1}$$

where H_0 represents the initial features of the inputs nodes, H_l the embeddings of the nodes at layer l and $\oplus$ denotes the residual connections (we add previous layers signals and apply normalization over the output). W_q, W_k and W_v are learnable matrices that are what we train to form the Queries, the Keys and the Values, in order 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.
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.
We use Multihead Attention as it is naturally fitted to treat unordered tokens,

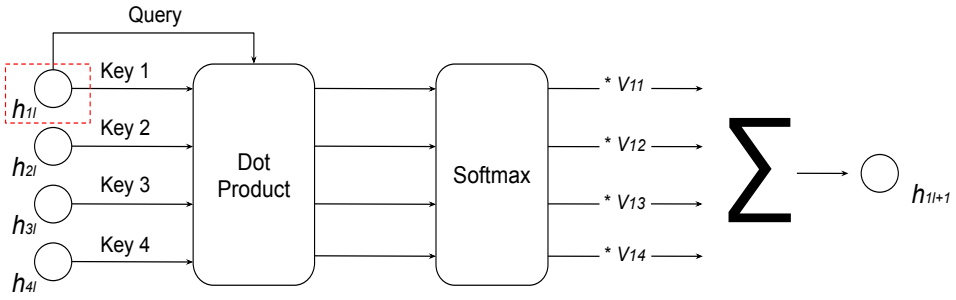

Fig. 2: Illustration of one Attention embedding layer applied to one node v_1 in a 4-node-network

and provides several good properties: it allows nodes to exchange information based on different criteria without necessarily needing edges as trustable mean of propagation, it can greatly avoid oversquashing (where long range dependencies tend to vanish as the number of message passing intermediates augment) and it can learn generalizable and intelligent gatherings of nodes' coordinates to create systematic patterns. Attention can also be prone to oversmoothing, particularly inside dense neighbourhoods of nodes, where the attention scores tend to explode and lead the nodes' embeddings to become similar, or get one node to be attended by or attending to an abnormally high number of other nodes, implying poor and altered embedding. Using skip connections can be a way to deal prevent some of these issues, as we will detail further on.
While this Attention-based message-passing scheme is appropriate, one major challenge remains: how can we infer links from these node representations ?

5 Our Attention-based solutions

5.1 Attention-only Model

One model we implement is an Neural Network that uses Attention both as the feature aggregator and for the prediction layer. Nodes exchange through Attention-based message passing as described in 3, and infer their predictions in the form of Attention scores. The prediction part of the network then only consists in such Attention scores computed by each node for each other node to go through unit-sized linear layers in order to output 0s or 1s, or to be directly

used as thresholded predictions. The computation of the attention scores, can then be formulated as:

$$E_{pred} = \text{softmax}\left(\frac{W_q H_l (W_k H_l)^T}{\sqrt{d_k}}\right)$$

We can also use unnormalized attention scores, which loses some of the benefits and "logistic" expressivity of the softmax, but allows greater range of values for link predictions, in order to get 0s and 1s more easily. In that case the formulation of the link predictions is simply:

$$E_{pred} = \frac{W_q H_l (W_k H_l)^T}{\sqrt{d_k}}$$

In practice, we use the softmaxed values as it allows better global coherence of the link predictions and generally create more plausible sets of links.

5.2 Graph Transformer

We hereby present a permutation-invariant structure that allows to infer a linkset without having to utilize a global, permutation-biased, Linear prediction layer as in [17]. The Graph Transformer, introduced in [14], can operate at both node and edge levels, which is an incredibly valuable property as it allows us to initialize edge objects and then transform them into proper links using learned "node-attentive" patterns. Since we don't have information on edges and do not want node-redundant information on the edges, we initialize them as simple neutral values (ones, in order not to disturb fist layers propagation) or noisy values following a normal distribution. In case where some communication links are impossible (physical obstacle, discretion needs, jamming...) we can initialize their value at 0, so that the model processes it as an impossible link. [19] to improve graph-level feature extraction. We add skip connections between layers, in order to stabilize gradients and help access previous layers' information. The principle of the model is to rely on the Attention computed between nodes to modulate the edge features. These edge features are also used to modulate the Attention-based signal propagation between nodes. Nodes and Edges are then updated interdependently and nodes serve as intermediates to create plausible edge patterns. We use residual connections, that prove to be important for deeper Neural Networks, and once again equip the model with Registers. We do not really use Positional Embeddings, that usually are materialized by Laplacian Eigenvectors [20][21] since we generally initialize our edges with ones and, even with random values, the positional encodings did not appear to make significant difference in the results, for a slightly more important computational cost due to the eigendecomposition.
The simplified embeddings of the set of nodes V going through one Graph Transformer layer l (for one Attention head, in practice we use up to 8) can be described as below:

$$H_{l+1} = \text{softmax}\left(\frac{W_q H_l (W_k H_l)^T}{\sqrt{d_k}} . W_e E_l\right) W_v H_l \oplus H_l \oplus H_{l-1}$$

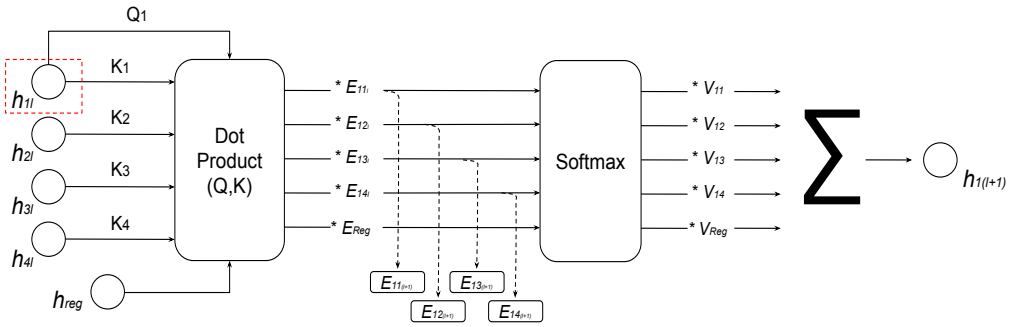

Fig. 3: Illustration of one Graph Transformer (dotprod attention) layer applied to one node v_1 in a 4-node-network

and

$$E_{l+1} = \frac{W_q H_l (W_k H_l)^T}{\sqrt{d_k}} . W_e E_l$$

with E_0 being a linear projection of the initial edge embeddings
We also implement a model which presents a similar structure but whose Attention is additive, as in [2]. In this case, the Attention between two nodes is computed by a 1-layer neural network that is fed the two nodes embeddings. We implement it in order to study whether it can help avoid some of the typical oversmoothing that appear in graph processing tasks, that the similarity-based retrieval formulation used by Dot-Product Attention does not totally avoid.

5.3 Disentangled Transformer

We propose a model whose the edge embeddings are a bit disentangled from the Attention computed between nodes. Here for an edge between i and j a small MLP is fed the concatenation of the i and j's embeddings to infer a edge-level embedding.

$$E_{ij} = tanh\left(W_e \begin{bmatrix} h_i \\ h_j \end{bmatrix} + b_e\right)$$

It then modulates the attention scores of the layers as in the standard Graph Transformer. We implement this component between Attention computation and the final edge embeddings, in order to add some flexibility to the edge embedding process. It is meant to alleviate some of the Attention tasks, since it has both to operate some information gathering message passing and to arrange Attention scores to form plausible edges. Here both are somewhat disentangled even though they are of course interdependant.
We also implemented a model that would only process the concatenated pairs of nodes and use Attention between the pairs, but it did not perform well as the predictions of the edges were almost totally independent, which means any

rather short edge would be predicted true with little to no respect to the other predicted edges or nodes. Indeed, the model did not seem to make great use of Attention, as we could observe when displaying the Attention scores and when trying the model with and without the Attention layer. This proves that the node-attention method to get edge predictions is much more expressive that the simple concatenation of the nodes' embeddings.

5.4 U-shaped Graph Transformer

The edge modulation process used by our Graph Transformer tends to make the node embedding process a bit lossy, especially if you consider the relatively simple and small input signal (nodes' positions): while it allows us to shape edges in a pleasant way, after several layers, the nodes can suffer from oversmoothing, or can have "derived" a bit. The learned edges are mostly "taught" to form probable links, and hence might not necessarily be perfectly suited for message passing, especially in the last layers, which are the results of many successive edge-modulated message passing. To address this issue, we connect last layers to the first ones in a U-Net [22] way. It is important to state that the model does not feature an downsampling-upsampling process, as U-Nets do (notably because graph upsampling is much more difficult than in an image processing context), the U-shape only comes from the use of skip connections between first and last layers as shown in 4. It enables the network to access once again the nodes' positions but, this time, with a almost formed set of links that bring a desirable bias that can guide the final prediction of the edges. We also implement it with the Attention-only model but it does not seem to be as relevant and important given the model size and the absence of edge modulation. For instance, the value of the nodes of the last layer would be:

$$H_{last} = \text{softmax}\left(\frac{W_q H_l (W_k H_l)^T}{\sqrt{d_k}} . W_e E_l\right) W_v H_l \oplus H_{last-1} \oplus H_0$$

5.5 Registers

As these edges are only modified via node-level Attention schemes, even if these can benefit from great expressivity and receptive field thanks to their consecutive node-to-node signal exchange, we wish to add a component aimed at improving global consistency by allowing to "store" graph-level attributes and patterns. We follow the guidelines established in [19] and adapt them to graphs. We add one or several dummy nodes whose initial embedding values are learned. These dummy nodes can then serve as registers, meaning that, through Attention-based message passing, they can be attended by the nodes of the graph to both store and retrieve information. It is useful to gather information that might not directly be related or useful to a single node or neighbourhood of nodes but, combined with many information signals from different nodes of the graphs it can aggregate in these registers as graph-level features. This type of global, long dependency gathering of information is generally a problem in graph message

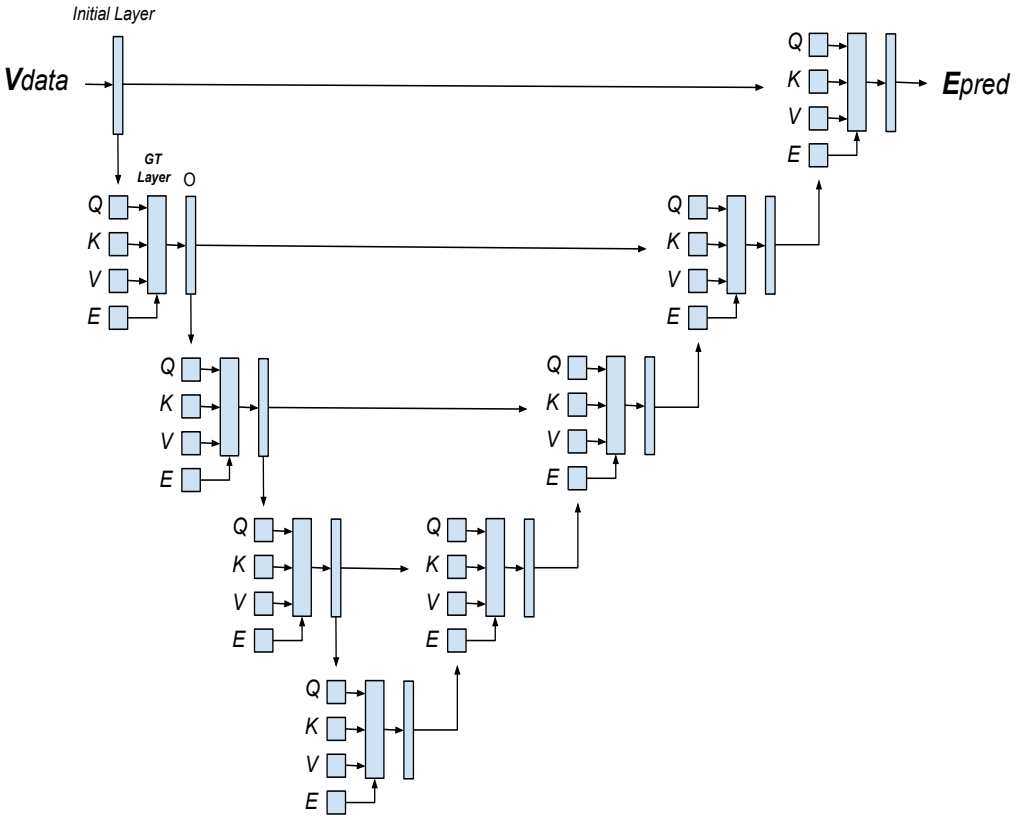

Fig. 4: Illustration of a simplified U-shaped Graph Transformer, first and last layers are directly connected

passing algorithms as nodes tend to keep neighbourhood-level interactions, as it suffers from oversquashing.

5.6 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 (Register or Cross-Attention), or using FiLM Layers [23][24] or a HyperNetwork [25] architecture. We then search F parametrized by θ such that

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

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. In the case where the attribute can be directly expressed as a differentiable function of the

output of the model and its weights, the respect of the desired the attribute in the predicted matrix can be enforced using Lagrangian Duality framework described in [26]. For instance, one could use an approximated differentiable expression of the connectedness or the total number of links or such, from the output of the model, and enforce it to be as close as their chosen value.

5.7 Generative formulation: Variational Inference

The problem itself is not deterministic, which might lead to sub-optimal training and results in the supervised learning setting: the output of the model, even if it corresponds to a possible solution for the given set of nodes, might be different than the exact output that was expected, and cause large loss values, even if the prediction was "valid" from a logical point of view. While this issue has not been extremely problematic in practice, since, while the problem is not deterministic, the dataset is (there are no equal sets of nodes' positions in the dataset), and the training seems rather stable. 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. This is a proper generative formulation of the problem that learns a continuous solution space, which can also be an interesting property to be able to explore the different possible solutions for a set of nodes. We then have a framework comparable with [12], without the need of matching algorithms on the output since our nodes are identified and initialized from the beginning of the decoding process as the generation is conditioned by the nodes' positions. The learning problem consists in feeding the true 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, typically GNN, followed by pooling layers in order to obtain a graph level encoded vector z. The decoder can be any of the previously described models (Attention model, Graph Transformer) described above but we add the encoded latent vector by either concatenating it with each node, or as a vector to be consulted through Attention.
The problem then results in maximizing the following:

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

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:

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

The idea that motivates this formulation is that, despite the strong signal brought by the position of the nodes, the variational encoder can still struggle to find the exact link topology corresponding to this set, especially if several solutions corresponding to a similar set of nodes lie in the dataset. In this case, the encoded vector, which brings additional information about the graph to be decoded, can help distinguish the exact solution we are looking for. Since it is continuous, the latent space in which this encoded vector lies can then be parsed to produce various 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. The training and the results did not happen to be much different than the supervised counterpart, notably because the nodes' feature space is rather large hence it is really unlikely to find similar sets of nodes in the dataset. Sampling through the latent space to feed the decoder different latent vectors for a same set of nodes still showed to produce different outputs, which means that the encoded vector is still somehow taken into account.

6 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 a few 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, results below are given for models trained for an equivalent of at most 20 epochs at 1e-4 learning rate. We generally use a standard binary cross entropy loss function. We use a dataset of multiple instances with randomly generated nodes' positions. It is composed of 180k samples of 16 nodes' positions and the adjacency matrix obtained solving the IP problem with a solver (namely Gurobi). Generating such a dataset is long and costly so we can not, for now, notably increase its size. To prevent overfitting, we wish to keep our models' number of parameters roughly in the same order of magnitude as our dataset size. Variational and supervised versions of the training and the models showed to display similar results and similar variations so we do not feature and distinguish both in the benchmarks.
We found that the Transformer-ish models sometimes tended to make conservative predictions between 0 and 0.4 instead of values closer to 0 and 1 so we added a sparsity enforcing penalty in the loss:

$penalty = \frac{|y|}{(\overline{p}-y)^2}$ with y being the prediction tensor and $\overline{p}$ an empirically observed mean for non-zero predicted values, in order to push predictions away from it (in practice it was close to 0.35). The Attention-only model showed to perform better with solely two rather large Attention layers, and no activation function, but we did not get it to output proper 0s and 1s without significant loss in relevance of the results. It would also output values rather continuously distributed from 0 to 0.4 so a simple 0.5 threshold would not work well, we then adaptively chose the threshold that offered the best accuracy, which in general was close to 0.25. The accuracy is measured as in a classical link prediction task

Table 1: Accuracy of both dataset and unseen graphs, results show that our models vasty outperform a baseline MLP and showcase the utility of some of the components.

Model	**Accuracy on dataset**	**Test accuracy**
Graph Transformer	94.70 %	88.68 %
2-layer Graph Transformer	93.55 %	87.48 %
U-shaped Graph Transformer	**95.04** %	**88.83** %
U-shaped Graph Transformer (additive attention)	94.69 %	88.79 %
U-shaped Disentangled Graph Transformer	94.78 %	88.67 %
Attention-only	94.49 %	88.59%
Flattened MLP (NOT permutation invariant)	89.86 %	84.73 %

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 1 contains almost 75% 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. We include the accuracy on a substract of the dataset (seen in training), because the task of imitating the samples is interesting in itself, to see how well the model can theoretically reproduce these graphs in the setting where they offer the best possible generalization property obtained for the test set (not seen in training or validation). We observe that models generally reproduce dataset graphs with high accuracy, Test set graphs, which the graphs have not seen during training, are not as well reproduced. This is not necessarily an issue since, as stated before, one set of nodes can admit several optimal solutions, and many more at least plausible solutions, which makes it really difficult to find the exact one that is expected. Both models still perform much better than the MLP baseline, and of course much better than random or only 0s guessing. The Attention-only model performs surprizingly well but requires an adaptive threshold. It happened to perform much better when using two Attention layers then using the Attention scores directly as outputs. The downside is that it can

not output values high enough for us to perform a simple 0.5 thesholding. Any more complex versions of the model would perform much more poorly. Overall, the best performing model is the Graph Transformer with the U-shaped skip connections. Graph Transformers globally show good performance and very few false negatives, but tend to be a bit greedy locally, predicting too many links.

Table 2: Verification of the range constraint on test graphs, Graph Transformers seem to produce more reliable links, results highlight the merit of the different components.

Model	**% of links of valid length**
Graph Transformer	96.3 %
2-layer Graph Transformer	94.05 %
U-shaped Graph Transformer	97.41 %
U-shaped Graph Transformer (additive attention)	97.27 %
U-shaped Disentangled Graph Transformer	**97.59** %
Attention-only	94.84 %
Flattened MLP (NOT permutation invariant)	85.37 %

We observe that, while Graph Transformers only beat the Attention model on the link prediction accuracy test by a small margin, they tend to output much more consistently valid links. We indeed observed that the Attention model could sometimes output too long hence unrealistic nodes while the Graph Transformers would rarely make such mistakes. We can deduct that the Attention-score prediction does well in terms of link patterns consistency, as the the number of predicted links is more realistic, globally and per node, but that the edge embedding process of Graph Transformers allow them finer control over the edges characteristics. Once again, the U-shaped Graph Transformers perform a bit better than the original Graph Transformer, and the ones that feature small MLPs to process the pairs of nodes seem to display slightly better results.
Since the aggregation mechanism is local, between nodes, the need to obtain complex, coherent patterns that involve high level view and combination of nodes' features requires the node embeddings to benefit from a large receptive field, which typically profits from models to be rather deep. In practice, we tested up to 10 Transformer-ish layers for each Graph-Transformer-like model and chose to feature 7 of them for most experiments. We empirically notice a significant increase of performance with the increase of the number of layers, while the simple Attention score prediction model does not really benefit from it. The process of "manipulating" the edges through the layers hence probably particularly enjoys deeper networks. Unfortunately, the deeper the models are (particularly when approaching 10 layers) the smaller the layers are, since we wish to keep the

number of parameters in the same order of magnitude as the dataset size, as mentioned before.

7 Conclusion

We tackled throughout this work the problem of infering optimized network topologies in constant time with a Deep Neural Network with strong theoretical generalization properties that is trained to imitate a dataset following a given algorithm (a Integer Programming solver in our case). We covered the notion of Permutation invariance and how important it is when dealing with graphs. We then showed how Attention is a perfect feature aggregator for such graph problematics and proposed several versions of Attention-based models from a pure Attention model using attention scores as predictions to Graph Transformers. Some additional architecture elements to Graph Transformers, namely U-shaped skip connections and Registers have been implemented to help us tackling our problem. We obtained very satisfactory results that highlight the utility of the components of our models. We hence provide a flexible, constant time model that provides trustable topologies of network links given the nodes' positions. It can greatly help instanteneously reaching some very important structural properties (connectedness...) and feature the desirable traits of the graphs that we train the model on. We highlighted the capacities of the different proposed architectures their contribution to the expressivity of the and the robustness of the predicted links. We hope our work has highlighted the interest of using Deep Learning methods to tackle combinatorial problems by turning them into monolithic global graph generation problems. Future steps will aim at improving the reliability of the infered topologies, possibly by introducing some link pruning algorithms. While this work is purely focused on one-shot generation of graphs, we will also investigate more progressive methods such as Denoising Diffusion Probabilistic Models [27], which follow similar "global generation" baselines but in an Langevin-dynamic-ish formulation.

Bibliography

[1] Su Yi, Yong Pei, and Shivkumar 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*, MobiHoc '03, pages 108–116, New York, NY, USA, June 2003. Association for Computing Machinery. ISBN 978-1-58113-684-5. https://doi.org/10.1145/778415.778429. URL https://doi.org/10.1145/778415.778429.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate, May 2016. URL http://arxiv.org/abs/1409.0473. arXiv:1409.0473 [cs, stat].

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, December 2017. URL http://arxiv.org/abs/1706.03762. arXiv:1706.03762 [cs].

[4] P. Gupta and P.R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000. ISSN 00189448. https://doi.org/10.1109/18.825799. URL http://ieeexplore.ieee.org/document/825799/.

[5] Amal Benhamiche, Wesley da Silva Coelho, and Nancy Perrot. Routing and Resource Assignment Problems in Future 5G Radio Access Networks. June 2019. https://doi.org/10.5441/002/inoc.2019.17.

[6] Wei Feng, Yong Li, Depeng Jin, Li Su, and Sheng Chen. Millimetre-Wave Backhaul for 5G Networks: Challenges and Solutions. *Sensors*, 16(6):892, June 2016. ISSN 1424-8220. https://doi.org/10.3390/s16060892. URL https://www.mdpi.com/1424-8220/16/6/892. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[7] Lichun Bao and J. J. Garcia-Luna-Aceves. Topology management in ad hoc networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*, MobiHoc '03, page 129–140, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581136846. https://doi.org/10.1145/778415.778432. URL https://doi.org/10.1145/778415.778432.

[8] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, October 2020. ISSN 1558-0008. https://doi.org/10.1109/JSAC.2020.3000405. Conference Name: IEEE Journal on Selected Areas in Communications.

[9] Kai Lei, Meng Qin, Bo Bai, Gong Zhang, and Min Yang. GCN-GAN: A Non-linear Temporal Link Prediction Model for Weighted Dynamic Networks, January 2019. URL http://arxiv.org/abs/1901.09165. arXiv:1901.09165 [cs].

[10] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, January 2009. ISSN 1941-0093. https://doi.org/10.1109/TNN.2008.2005605. Conference Name: IEEE Transactions on Neural Networks.

[11] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, July 2017. ISSN 1053-5888, 1558-0792. https://doi.org/10.1109/MSP.2017.2693418. URL http://arxiv.org/abs/1611.08097. arXiv:1611.08097 [cs].

[12] Martin Simonovsky and Nikos Komodakis. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders, February 2018. URL http://arxiv.org/abs/1802.03480. arXiv:1802.03480 [cs].

[13] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[14] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs, 2021.

[15] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks, 2020.

[16] Waiss Azizian and marc lelarge. Expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=lxHgXYN4bwl.

[17] Félix Marcoccia, Cédric Adjih, and Paul Mühlethaler. A deep learning approach to topology configuration in multi-hop wireless networks with directional antennas: nodes2net. In *2023 12th IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*, pages 1–6, 2023. https://doi.org/10.23919/PEMWN58813.2023.10304906.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, December 2015. URL http://arxiv.org/abs/1512.03385. arXiv:1512.03385 [cs].

[19] Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers, 2023.

[20] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003. https://doi.org/10.1162/089976603321780317.

[21] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks, 2022.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[23] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer, 2017.

[24] Marc Brockschmidt. Gnn-film: Graph neural networks with feature-wise linear modulation, 2020.

[25] David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks, 2016.

[26] Ferdinando Fioretto, Pascal Van Hentenryck, Terrence WK Mak, Cuong Tran, Federico Baldo, and Michele Lombardi. Lagrangian duality for constrained deep learning, 2020.

[27] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.